VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY
INSTITUTE OF INFORMATICS

VÝVOJ WEBOVÉ APLIKACE

WEB APPLICATION DEVELOPMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Šimon Procházka
AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Dydowicz, Ph.D.
SUPERVISOR

BRNO 2019

Zadání bakalářské práce

Ústav:	Ústav informatiky
Student:	**Šimon Procházka**
Studijní program:	Systémové inženýrství a informatika
Studijní obor:	Manažerská informatika
Vedoucí práce:	**Ing. Petr Dydowicz, Ph.D.**
Akademický rok:	2018/19

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává bakalářskou práci s názvem:

Vývoj webové aplikace

Charakteristika problematiky úkolu:
Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrh řešení, přínos práce
Závěr
Seznam použité literatury

Cíle, kterých má být dosaženo:

Vývoj webové aplikace pro vytváření frameworkové dokumentace společnosti LOGEX Solution Center s.r.o.

Základní literární prameny:

BASL, J. a R. BLAŽÍČEK. Podnikové informační systémy. Podnik v informační společnosti. Praha: Grada, 2008. 283 s. ISBN 978-80-247-2279-5.

MOLNÁR, Z. Automatizované informační systémy. Praha: Strojní fakulta ČVUT, 2000. 126 s. ISBN 80-01-02269-2.

MOLNÁR, Z. Efektivnost informačních systémů. Praha: Grada Publishing, 2000. 142 s. ISBN 80-716--410-X.

ŘEPA, V. Analýza a návrh informačních systémů. Praha: Ekopress, 1999. 403 s. ISBN 80-86119-- 3-0.

SODOMKA, P. a H. KLČOVÁ. Informační systémy v podnikové praxi. Brno: Computer Press, 2010. 501 s. ISBN 978-80-251-2878-7.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2018/19

V Brně dne 28.2.2019

L. S.

doc. RNDr. Bedřich Půža, CSc.	doc. Ing. et Ing. Stanislav Škapa, Ph.D.
ředitel	děkan

Abstrakt

Bakalářská práce je zaměřena na vývoj webové aplikace, která bude sloužit jako dokumentační nástroj pro framework společnosti LOGEX Solution Center s.r.o. Tato aplikace byla vytvořena za využití frameworku Angular a byla napsána v jazyce TypeScript za využití HTML a CSS.

Klíčová Slova

Webová aplikace, Angular, JavaScript, TypeScript, HTML, CSS, LOGEX

Abstract

The topic of this bachelor's thesis is the development of a web application which will be used as a documentation tool for the framework of the LOGEX Solution Center s.r.o. company. The application was created with TypeScript, HTML and CSS as well as the Angular framework.

Key words

Web application, Angular, JavaScript, TypeScript, HTML, CSS, LOGEX

Bibliografické citace

PROCHÁZKA, Šimon. *Vývoj webové aplikace* [online]. Brno, 2019 [cit. 2019-04-28]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/118383. Bakalářská práce. Vysoké učení technické v Brně, Fakulta podnikatelská, Ústav informatiky. Vedoucí práce Petr Dydowicz.

Čestné prohlášení

Prohlašuji, že předložená bakalářská práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 6. 5. 2019

……...

podpis autora

Poděkování

Rád bych touto formou poděkoval vedoucímu mé bakalářské práce Ing. Petru Dydowiczi, PhD. za vedení bakalářské práce a čas, který mi věnoval. Dále bych rád poděkoval společnosti LOGEX Solution Center s.r.o. a jejím zaměstnancům za aktivní přístup a pomoc při zpracovávání práce.

OBSAH

Úvod

V současné době má technologie exponenciální růst, a vliv, který mají počítače na náš každodenní život, stoupá nejméně stejnou rychlostí. Subjektem tohoto růstu jsou také webové stránky a technologie, kterými se vytváří. Zatímco dříve byly webové stránky spíše zaměřeny na vzhledovou část a vývojáři často obstarávali role designera i programátora, dnes je vývoj webu náročný i z hlediska funkčnosti a vývojář zřídkakdy zaskakuje všechny role. Ne každý programátor má totiž zkušenosti s UI/UX, a ne každý designer umí programovat.

Ve své bakalářské práci se zabývám návrhem a vývojem webové dokumentace pro firmu LOGEX Solution Center s.r.o. Tato firma již má jednu demonstrační stránku, v současné době je však tato stránka vzhledově i funkčně nedostačující.

CÍLE PRÁCE, METODY A POSTUPY ZPRACOVÁNÍ

Hlavním cílem této bakalářské práce je návrh a vývoj webové dokumentace pro firmu LOGEX Solution Center s.r.o. Současná demonstrační stránka není pro vývojáře dostačující, a pokud se chtějí podívat na využití a možnosti manipulace konkrétních částí frameworku, musí nejprve otevřít dokument a hledat tyto informace uvnitř kódu.

Práce také zohledňuje požadavky, které mi sdělili zaměstnanci firmy. S nimi jsem též konzultoval možné přístupy. Práce musí být napsána v jazyce TypeScript za využití frameworku Angular. Dále musí být jednoduché přidávání komponent do dokumentace a možnost zobrazení zdrojového kódu konkrétního příkladu.

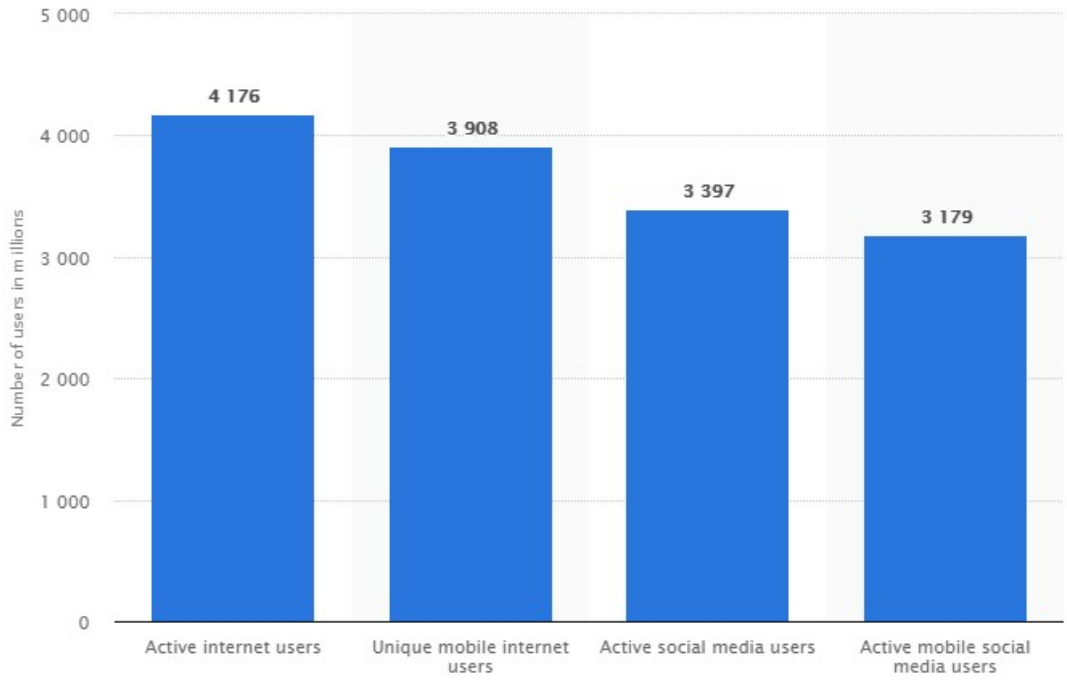
Bakalářská práce využívá v analytické části SWOT analýzy, pomocí kterých jsem se blíže podíval na různé dokumentace. Na základě výsledku analýz jsem navrhl řešení a napsal kód, který může obsahovat tuto dokumentaci.

1 TEORETICKÁ VÝCHODISKA PRÁCE

V zájmu přiblížení problematiky vývoje webových stránek v této kapitole popíši teoretická východiska, na kterých je práce postavena.

1.1 Webová aplikace

Teoretickou část zahájím vysvětlením samotného významu webových aplikací. Jedná se o počítačový program, který umožňuje provádět různé úkony za využití internetu. Tento program se často vyznačuje tím, že umožňuje uživateli odkazovat se na server a čerpat data z databáze (1). Rostoucí dostupnost internetu a možnost se k němu připojit prakticky z jakéhokoliv místa má za následek rostoucí dobu strávenou uživateli na těchto stránkách a vyšší počet uživatelů (2). Na grafu č. 1 lze vidět počet aktivních uživatelů internetu, který již přesáhl hranici 4 miliard. Velmi vysoko jsou také postaveny uživatelé mobilních telefonů, které jsou těsně pod touto hranicí. Sociální média jsou také hojně používané s více než třemi miliardami uživatelů (3). To jsou důvody, proč jsou webové aplikace velmi populární a žádané.



Graf č. 1: Počet uživatelů (3)

1.1.1 Prohlížeče

Prohlížeč slouží k zobrazování webových stránek a procházení obsahu webu pomocí hypertextových odkazů. (4)

Existuje mnoho různých prohlížečů (5; 6; 7; 8). Jejich základní prvky jsou stejné, ale některé příkazy a funkce webových stránek mohou být v různých prohlížečích interpretovány odlišně s jiným výstupem (9). Při vývoji webové aplikace je důležité mít tuto skutečnost na paměti a snažit se zajistit, aby aplikace podporovala co největší množství prohlížečů – zejména pak ty, které používá cílová skupina uživatelů aplikace.

Existují také statistiky, které se zabývají podílem různých prohlížečů na trhu. Některé stránky, jako třeba W3Schools, sbírají data na základě návštěvnosti na vlastních stránkách. Zatímco W3Schools udává 80% využívání prohlížeče Chrome (10), data od GlobalStats a Net Applications udávají přibližně 60 % (11; 12). Dá se tedy konstatovat, že prohlížeč Google Chrome je nejčastěji používaným prohlížečem.

1.1.2 Document Object Model

Pravděpodobně nejdůležitější částí webových stránek je Document Object Model. DOM představuje dokument jako logický strom, který na konci svých větví obsahuje uzly. Tyto uzly nakonec obsahují samostatné objekty, ze kterých se skládá webová stránka. DOM umožňuje programově přistupovat k tomuto stromu a měnit tak strukturu a vzhled dokumentu (13). Pokud chceme HTML prvky modifikovat, je nutné odkázat se na DOM a v něm najít konkrétní prvek (element). Jelikož se jedná o pomalý proces, některé aplikace a rozhraní využívají tzv. Virtual DOM, který drží logiku uvnitř kódu a dělá aplikaci rychlejší (14). O rozhraních se dozvíme více v kapitole 1.4.

1.1.3 Responzivní design

Rozmach mobilních zařízení si vyžádal vývin technologií responzivního designu, které umožňují efektivní a přehledné zobrazování webových stránek i na malých obrazovkách. Chování stránek s responzivním designem se totiž mění podle velikosti obrazovky zařízení, ve kterém jsou spouštěny. Práce se této problematice nevěnuje, protože vyvíjená aplikace je určena především pro větší obrazovky. Přesto je vhodné alespoň zmínit, že možnosti responzivního designu existují a hojně se využívají.

Nejznámějším a nejvíce využívaným nástrojem pro responzivní design je rozhraní Bootstrap. (15)

1.2 Technologie pro vývoj webových aplikací

Při vývoji webových stránek se využívají technologie, bez kterých webová stránka nemůže existovat. V této podkapitole se zabývám popisem těchto technologií.

1.2.1 HTML

Hypertext Markup Language je značkovací jazyk, který umožňuje tvorbu objektů na stránce. Jednotlivé prvky jsou vytvořeny za využití značek, které mohou být jak párové, tak samotné. Na obrázku 1 je ukázka párových značek v základním zápise HTML dokumentu (16).

```
<!DOCTYPE html>
<html>
<title>HTML Tutorial</title>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Obrázek č. 1: HTML příklad: Párové značky (Zdroj: 16)

Od roku 2014 je nejnovější verze HTML5, jejíž nové prvky umožňují používání řady nových funkcí. Díky novým grafickým elementům <svg> je možné vytvářet vektorovou grafiku přímo na webu (16). Je tak možné grafy pro dokumenty založené na prezentaci dat. Mohly tak vzniknout knihovny jako D3.js, které slouží k vytváření takových grafů (17). Přestože HTML5 je aktuální od roku 2014, definici prvku SVG můžeme najít již v roce 2011 (18).

Dalším zajímavým prvkem je <canvas>. Prvky *canvas* mohou být interaktivní, animované a můžeme je obecně využít ke kreslení. Tento prvek můžeme lze také použít pro vytváření her na webu (19). Knihovna WebGL (Web Graphics Library), která umožňuje vykreslení 2D a 3D grafiky na webu, ke své funkci využívá hlavně *canvas* (20).

Unity, nástroj k vytváření her za využití jazyka C#, umožňuje kompilaci kódu na JavaScript a využívá tuto knihovnu pro jeho vykreslení (21).

1.2.2 CSS

CSS (Cascading Style Sheets), slouží k úpravě zobrazení HTML elementů a umožňuje nám měnit vzhled dokumentu (22). Poslední verze kaskádových stylů CSS3 umožňuje vytvářet také animace (23) či různé zobrazení pro různé velikosti obrazovek – responzivní design (24). Tuto funkcionalitu CSS3 velmi využívá rozhraní Bootstrap (15).

1.2.2.1 SCSS

SCSS (Sassy CSS) je supersetem CSS. Kód, který je validní v CSS, je validní také v SCSS. SCSS umožňuje snáze vytvářet specifické styly pomocí jejich zanořování. Abychom mohli využít tento způsob zápisu, před spuštěním programu je nutné zkompilovat tyto soubory na CSS (25).

Kromě zanořování stylů nám SCSS umožňuje také mít proměnné, rozšířit již napsané styly pomocí klíčového slova *@extend*, nebo také numerické operace (26).

```scss
.class-one {
    margin-top: 10px;
    &.class-one-extending-style {
        padding-top: 10px;
    }
    .inner-class {
        background-color: blue;
    }
}
```

```css
.class-one {
    margin-top: 10px;
}

.class-one.class-one-extending-style {
    padding-top: 10px;
}

.class-one .inner-class {
    background-color: blue;
}
```

Obrázek 2: SCSS a CSS (Zdroj: Vlastní zpracování)

Na obrázku č. 2 lze vidět dva různé kódy, které mají stejný výstup. Na pravé straně lze vidět kód v CSS, ve kterém je nutné vícekrát napsat stejný styl, zatímco v SCSS (levá strana) můžeme další styly vnořit. Kód je potom čitelnější, kratší a je snazší si jej představit.

1.2.3 JavaScript

JavaScript, oficiálním názvem ECMAScript (27), je jedno-vláknový (28), objektově orientovaný programovací jazyk (29). JavaScript je jedním ze tří hlavních jazyků webu (další dva jsou již zmíněné HTML a CSS). Obvykle JavaScript provádí kód

v prohlížeči, tedy na straně uživatele, jsou však nástroje, které umožňují jeho využití jinde (30).

Mezi takové využití patří například Node.js, který umožňuje psaní serverových aplikací v JavaScriptu a využívá asynchronní operace pro maximalizaci výkonu. Node.js pracuje na JavaScriptovém enginu V8 (30) a je napsaný v C++ (31). Node.js se také využívá pro práci s Angularem (32).

JavaScript zaštiťuje na webu možnost reagovat na uživatele, jinak by stránky byly statické. Umožňuje měnit HTML prvky, jejich obsah a vzhled, ale lze díky němu také vytvářet nové prvky na stránce. Díky tomu můžeme reagovat na uživatele a pracovat s jeho požadavkem (29).

Již zmíněné prvky HTML5 se velmi využívají ve vývoji. Knihovna D3.js umožňuje nejen vytvářet grafy z dat, které máme k dispozici, ale také reagovat na pohyb myší a zvýraznit konkrétní místo na grafu na základě pozice myši (33).

S JavaScriptem se dá také zabývat strojovým učením. Knihovna TensorFlow.js je knihovna pro strojové učení, která využívá Node.js a umožňuje nám pracovat na programech, ve kterých se stroj sám učí, přímo v prohlížeči (34).

V JavaScriptu je také napsáno mnoho desktopových aplikací za využití frameworku Electron. Mezi známé aplikace patří například vývojové prostředí Visual Studio Code, chatovací aplikace Slack a Discord, nebo také Skype (35).

1.2.4 TypeScript

V roce 2012 přišla společnost Microsoft s TypeScriptem. Microsoft o TS tvrdí, že je otypovaným supersetem JS, který se kompiluje do samotného JavaScriptu. Protože prohlížeče využívají právě JS pro funkcionalitu, TS je nutné nejprve zkompilovat (transpilovat) na JavaScriptový kód. Jakýkoliv JavaScriptový kód je validní v TypeScriptu, je tedy možné vytvořit JavaScriptový program a pouze změnit jeho koncovku, abychom docílili validního TypeScriptového programu (36).

Jak již název napovídá, výhodou TypeScriptu jsou otypované proměnné, díky kterým je program více čitelný a předvídatelný. Velká část vývojových prostředí (IDE) umožňuje statickou kontrolu typů, a v případě, že typy nesedí, vývojář je informován o

této chybě. To se může stát v případě, že vývojář se pokusí vložit textovou hodnotu do číselné proměnné (37).

1.2.5 Transpilace

Jak jsem již zmínil, TypeScript musí být nejprve kompilovaný do JavaScriptu, aby mohl běžet v prohlížeči. Tomuto procesu se říká transpilace a nejznámějším nástrojem je Babel (38). I JavaScript je občas nutné transpilovat. Je to z toho důvodu, že některé nejnovější funkce JavaScriptu nemusí být v prohlížečích podporovány (39). Z toho důvodu se využívají transpilery při vývoji webových aplikací – překládají kód jiného jazyka, nebo novějších verzí JavaScriptu do starších, prohlížečem podporovaných, funkcí (40).

Pro kompatibilitu starších prohlížečů se nevyužívají pouze transpilátory. Tzv. polyfilly se také mohou použít k implementaci nových, nepodporovaných funkcí, za využití starého kódu. Polyfilly nejčastěji najdeme spjaté s konkrétním API prohlížečů. Zatímco transpilery mění celý kód, polyfilly slouží spíše pro kompatibilitu konkrétní části kódu (40).

```
class BaseClass {
}

class Level1 extends BaseClass {

}

class Level2 extends Level1 {

}
```

```
var __extends = (function () {
    var extendStatics = function (d, b) {
        extendStatics = function (d, b) { for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p]; };
        return extendStatics(d, b);
    }
    return function (d, b) {
        extendStatics(d, b);
        function __() { this.constructor = d; }
        d.prototype = (__.prototype = b.prototype, new __());
    };
})();
var BaseClass = /** @class */ (function () {
    function BaseClass() {
        this.a = 1;
    }
    return BaseClass;
}());
var Level1 = /** @class */ (function (_super) {
    __extends(Level1, _super);
    function Level1() {
        return _super !== null && _super.apply(this, arguments) || this;
    }
    return Level1;
}(BaseClass));
var Level2 = /** @class */ (function (_super) {
    __extends(Level2, _super);
    function Level2() {
        return _super !== null && _super.apply(this, arguments) || this;
    }
    return Level2;
}(Level1));
```

Obrázek č. 3: Transpilovaný TS kód (Zdroj: Vlastní zpracování)

Na obrázku č. 3 je ukázka transpilovaného TS kódu za využití příkazu *tsc*. Kód na obou stranách provede identickou činnost, avšak na pravé straně je převedený do formy přijatelné pro prohlížeče.

1.3 Synchronní a asynchronní programování

Jak jsem již zmínil v kapitole o JavaScriptu, v programování existuje pojem asynchronní operace. Je nutné podotknout, že následující struktury a modely jsou implementovány v rámci prohlížeče, což má za následek rozdíly mezi prohlížeči. Existuje mnoho videí a nástrojů, které vizuálně zobrazují, co se děje při asynchronní manipulaci s úkoly (41; 42).

1.3.1 Task queue

Task queue je struktura, která udává pořadí, v jakém se mají funkce vykonat. Když spustíme program, na začátku task queue najdeme funkci, která se objevila v kódu nejdříve. Tato funkce je dále poslána do event loop (41).

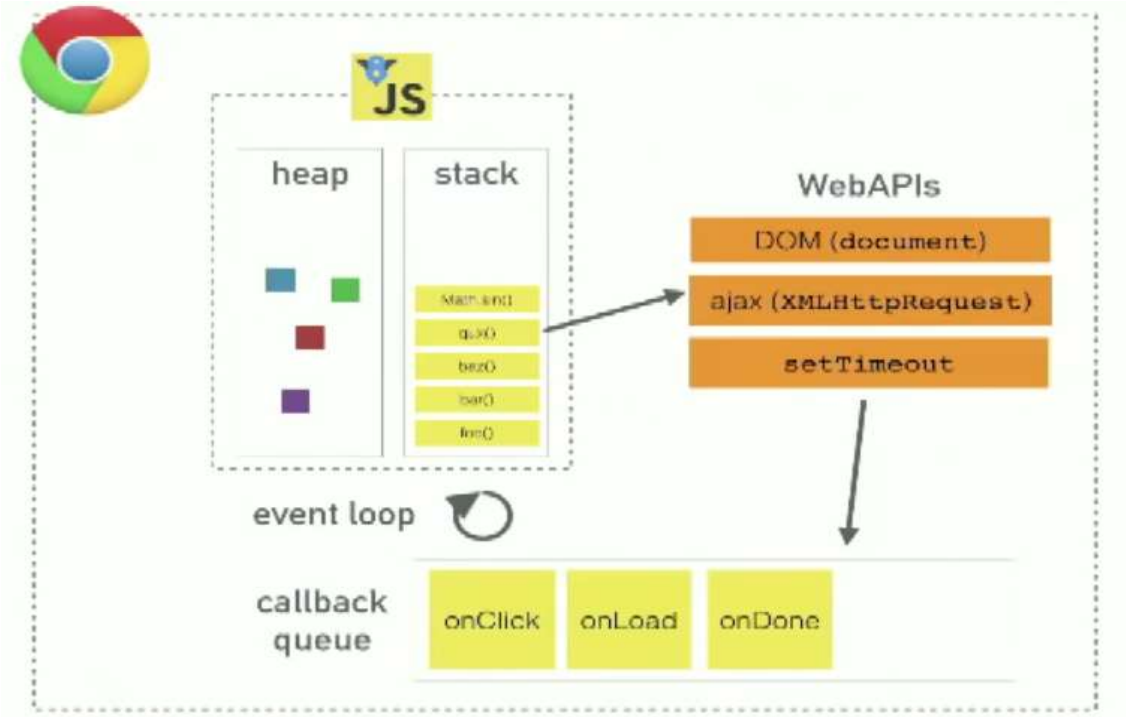
1.3.2 Event loop

Model, v němž se vykonávají funkce z Task queue, se nazývá Event loop. Jelikož JavaScript může dělat pouze jednu operaci, v momentě, kdy nastane dlouhotrvající operace, tato event loop se zasekne a neposune se dál, dokud nedokončí poslední operaci. V případě nekonečné smyčky to znamená, že event loop nikdy nedokončí svoji současnou funkci a nedostanou se tak do ní další úkoly z task queue (41).

1.3.3 Call stack

Když se zavolá funkce, vytvoří se call stack. Call stack je datová struktura, která definuje, kde v programu se zrovna nacházíme. Jelikož se jedná o zásobník, poslední volaná funkce se vždy provede jako první. V momentě, kdy se zavolá nějaká funkce, přidá se do call stacku. Poté, co funkce vrátí hodnotu, zmízí z call stacku (42).

Spolupráci těchto struktur je možné vidět na obrázku č. 4, který mimo jiné ukazuje, že call stack je jediná struktura implementovaná v JavaScriptu.



Obrázek č. 4: Spolupráce jednotlivých struktur (Zdroj: 43)

1.3.4 Synchronní programování

Na základě popsaných struktur vidíme, že JavaScript nemůže provádět více operací ve stejný moment. Jelikož má pouze hlavní vlákno, může provádět v jeden moment pouze jednu operaci. V případě, že se do event loop dostane nekonečná smyčka, bude se provádět pouze tato operace a všechny další operace se pošlou do fronty, která se nikdy nevyprázdní. V případě, že tato situace nastane, prohlížeč se zastaví a neumožní uživateli žádnou další operaci, neboť se stále pokouší dokončit současnou (44).

1.3.5 Asynchronní programování

Asynchronní programování v samotném JS není možné. Můžeme toho však dosáhnout za využití API prohlížeče. Application Programming Interface (API) můžeme najít v téměř každém programu. Jedná se o veřejnou, často jednoduchou část programu, kterou může vývojář použít bez hlubší znalosti tohoto programu (45).

V případě webového vývoje se můžeme odkazovat na API prohlížeče nebo API třetích stran. API prohlížeče můžeme využít například pro dotazy na data, která jsou uložená na serveru (46). API třetích stran pak může být využito pro získání často vyhledávaných slov na sociálních sítích, z kterých pak můžeme dělat analýzy (47).

```
console.log('This is the first log in console');

setTimeout( () => {
  console.log('This is the third log in console');
}, 0);       You, a few seconds ago • Uncommitted chan

console.log('This is the second log in console');
```

Obrázek č. 5: Příklad na asynchronní funkci (Zdroj: Vlastní zpracování)

Na obrázku č. 5 je k nahlédnutí kód, který obsahuje výpisy do konzole. Druhý výpis v pořadí se však vypíše jako poslední. Je to z toho důvodu, že funkce setTimeout využívá API globální proměnné *window*, tedy objektu obsahující dokument (48). Jedním z parametrů funkce je další funkce (tyv. callback), který určuje, co se má stát po uběhnutí časového intervalu. Jelikož tato funkce nepracuje na hlavním vlákně JavaScriptu, mohou tyto dvě operace probíhat zároveň. Jakmile čas vyprší, funkce se přidá do task queue a zavolá se jako první po dokončení všech předchozích operací (44). V konzoli nalezneme výstup, který lze vidět na obrázku č. 6.

```
This is the first log in console
This is the second log in console
This is the third log in console
```

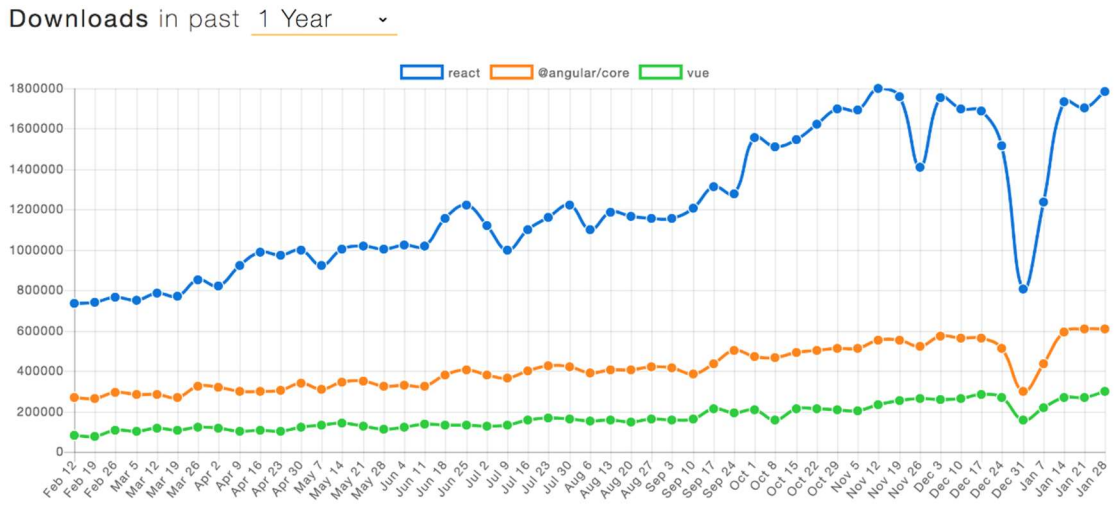
Obrázek č. 6: Výsledek z příkladu na obrázku č. 5 (Zdroj: Vlastní zpracování)

Je důležité podotknout, že každý prohlížeč pracuje se svým API rozdílně, je tedy možné, že asynchronní program spuštěný v prohlížeči Chrome může mít jiný výsledek než v prohlížeči Internet Explorer (49).

Asynchronní programování je důležité zejména při snaze urychlit stránku. Existuje mnoho nástrojů, které slouží ke zhodnocení rychlosti načítání stránek (50; 51). Tyto testy jsou velice důležité, protože rychlejší stránky představují vyšší počet návštěvníků a jsou lépe hodnoceny vyhledávači. Pomalým stránkám hrozí, že uživatelé odejdou za konkurencí. Pokud se stránka načítá déle než 3 sekundy, má v průměru o 22 % méně návštěv než stránka, která se načte během první vteřiny (52).

1.4 Další nástroje pro vývoj webových aplikací

Pro vývoj webových aplikací existuje mnoho nástrojů, které můžeme využít. Mezi tyto nástroje patří knihovny či frameworky (rozhraní). Mezi třemi nejčastějšími frameworky najdeme Angular, React a Vue (53). Z toho důvodu se v této části zaměřím zejména na tyto frameworky, ale zmíním také další nástroje, které v práci využívám. Statistiku nejvíce používaných frameworků lze najít na grafu č. 2, který udává, že React je nejvíce používaným z nich.



Graf č. 2: Nejvíce využívané frameworky (Zdroj: 53)

1.4.1 Angular

Framework Angular můžeme rozdělit na dvě části. První částí je AngularJS, který vznikl v roce 2010 a je udržován zejména společností Google, (54) nicméně projekt je open-source a jiní vývojáři se mohou stát podílet na funkcionalitách AngularJS. (55) Druhou části je Angular 2, který vznikl v roce 2016 (56), a přichází s ním také Command Line Interface (CLI), tedy příkazová řádka, pomocí které můžeme rychleji vytvářet jednotlivé části naší webové aplikace (57).

Angular je velmi zaměřený na znovu použitelný kód. Využívá komponenty obsahující šablony pro psaní HTML kódu, případně kaskádových stylů pro konkrétní šablony, a tyto komponenty lze použít ve stejné podobě na více místech. Komponenty mohou být také vytvořeny dynamicky s jinými vlastnostmi. Díky této dynamičnosti

můžeme mít dvě, v kódu identické komponenty na jedné stránce, avšak každá bude zobrazovat něco jiného (58).

1.4.1.1 Angular Material

Angular Material je knihovna obsahující moduly pro kvalitní uživatelské rozhraní (59). Za vývojem komponentů z této knihovny stojí vývojáři samotného Angularu (60). V bakalářské práci využívám nějaké komponenty z Angular Material, které nám usnadní dynamické vytváření komponent.

1.4.2 React

React je JavaScriptová knihovna z roku 2013, která je udržována zejména Facebookem, ale stejně jako Angular je open-source a mohou do ní přispět i ostatní lidé (61). Využívá JSX, respektive syntaxového rozšíření JavaScriptu (JavaScript eXtension), které umožňuje například mít HTML prvky jako proměnnou. Zatímco v jednoduchých webových stránkách prohlížeč vytvoří DOM stránky (62), React využívá JavaScriptu namísto šablon a umožňuje udržovat logiku stránky abstraktně, tedy jako data ve virtuálním DOMu (63). Aplikace je díky tomu rychlejší, neboť se nemusí stále odkazovat na DOM, a je snazší udržovat stav aplikace v kódu. Na stránce se komponenta promítne až po funkci render(), která se projeví na samotném DOMu (14).

1.4.3 Vue.js

Třetí z řady nejvíce používaných nástrojů pro tvorbu webů je Vue.js. Tento framework byl vytvořen v roce 2013 Evanem You a také se jedná o open-source projekt (64). Vue využívá také komponenty, které mohou být dynamicky měněny, využívá šablon podobně jako Angular, avšak stejně jako React využívá Virtual DOM (65).

1.4.4 Material Design

Material Design obsahuje fonty a další designové prvky, které usnadňují práci při vývoji webu. Podobně jako Angular Material, i tento projekt zaštiťuje Google a je open-source. Díky tomuto projektu můžeme snadno a rychle vytvořit uživatelsky přívětivé stránky (66). V rámci bakalářské práce využívám Material Design například v navigaci, kde jsou použity nějaké ikony z tohoto projektu.

1.4.5 HighlightJS

V bakalářské práci vytvářím dokumentační aplikaci. Jelikož se jedná o dokumentaci, která bude obsahovat příklady, je vhodné, aby tyto příklady zobrazovali také zdrojový kód. K tomu účelu používám knihovnu HighlightJS, kterou původně vytvořil Ivan Sagalaev v roce 2006. Přestože na tomto projektu již nepracuje, projekt je udržovaný komunitou. Knihovna slouží k vyznačování syntaxe různých programovacích jazyků v JavaScriptu (67; 68).

1.4.6 Wireframing

Wireframing je způsob návrhu webu, který slouží k základním návrhům vzhledu a funkce stránky. Přestože existují nástroje, které usnadňují tvorbu takového návrhu, wireframing může mít podobu také jednoduchého náčrtu na papíru (69).

1.4.7 Compodoc

Compodoc je dokumentační nástroj pro aplikace napsané v Angularu. Tento nástroj přichází s rozhraním příkazové řádky, díky které můžeme naši aplikaci zdokumentovat automaticky. Při dokumentaci prochází obsah každého souboru, které seskupí na základě využití v modulech či složkách a vygeneruje webovou stránku obsahující všechny proměnné a funkce, které komponenta využívá. Při využití tohoto nástroje je možné automaticky vygenerovat dokumentaci pro celý projekt (69).

Pro compodoc je možné specifikovat, jaké proměnné a funkce mají být zdokumentovány. Je tak možné například zakázat dokumentaci pro proměnné či funkce, pro které je zakázaný přístup mimo komponentu (70).

1.5 Analytické nástroje

Jelikož nezbytnou součástí bakalářské práce je analýza, je nutné využít vhodné analytické nástroje a modely, které nám pomohou více pochopit problematiku a usnadnit nám tvorbu práce. V této kapitole se zabývám jednotlivými analytickými nástroji, které jsou v práci využity.

1.5.1 SWOT Analýza

Strengths, Weaknesses, Opportunities and Threats, zkráceně SWOT analýza, je nástroj k ohodnocení subjektu. Přestože tento subjekt často bývá společnost, je možné využít tuto analýzu k ohodnocení místa, oboru či konkrétního projektu (71).

Tuto analýzu můžeme rozdělit na dvě části. První část jsou vnitřní vlivy, mezi které patří silné a slabé stránky subjektu. Součástí vnějších vlivů jsou potom hrozby a příležitosti, které se zabývají spíše prostředím, v němž subjekt figuruje (71).

Jinak řečeno, silné stránky nám říkají, co náš subjekt dělá dobře. Díky této analýze jsme schopní vyjádřit, v čem máme náskok oproti jiným, podobným subjektům. Slabé stránky naopak ukazují, co bychom měli dělat lépe či jakým postupům bychom se měli vyhnout. Příležitosti nám mohou přiblížit, co se v současnosti děje v okolí. Mohou nás směrovat ke krokům, které můžeme podniknout pro vylepšení našeho subjektu. Hrozby naopak ukazují různé překážky v dosáhnutí cíle, případně jiné změny, které mohou ohrozit náš projekt (71; 72).

SWOT analýza má zpravidla rozdělení do tabulky o dvou sloupcích a dvou řádcích. První řádek vyjadřuje silné a slabé stránky, respektive vnitřní vlivy na subjekt. Druhý řádek pak obsahuje vnější vlivy. První sloupec také vyjadřuje prospěšné vlivy, zatímco druhý sloupec ukazuje prvky, které mohou našemu subjektu škodit (71). Strukturu SWOT analýzy lze také vidět na obrázku č. 7.

Obrázek č. 7: SWOT Analýza (Zdroj: 71**)**

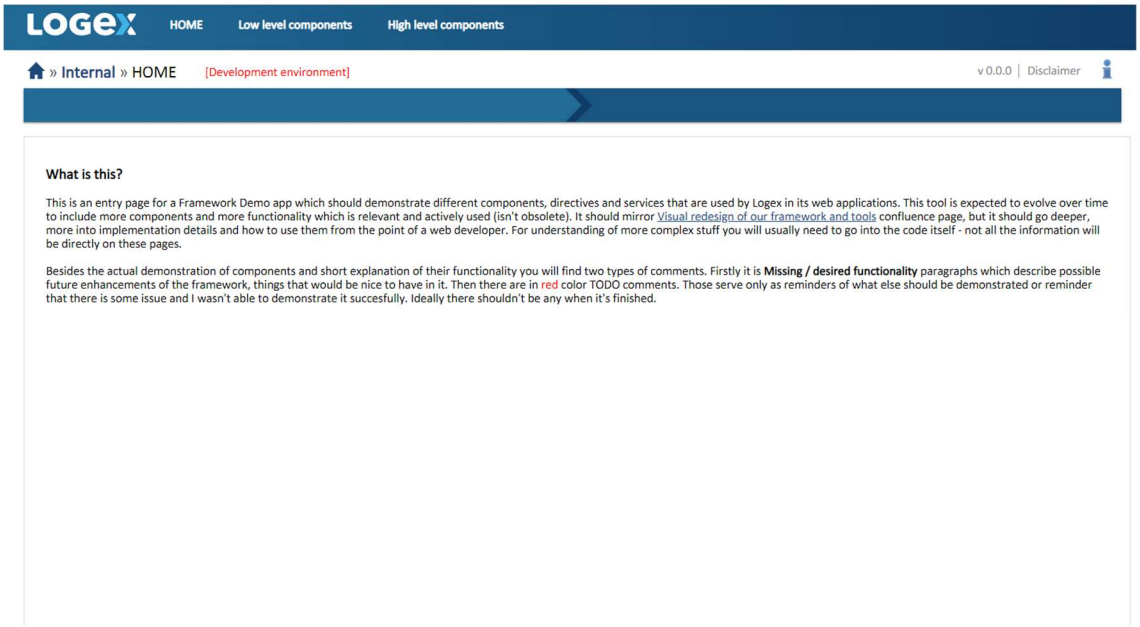
2 ANALÝZA SOUČASNÉHO STAVU

Tato kapitola pojednává o současném stavu dokumentace v rámci firmy LOGEX Healthcare Analytics, pro jejichž vývojové středisko je tato bakalářská práce zpracována. V analýze se také zabývám dalšími dokumentacemi či dokumentačními nástroji, ze kterých je možné čerpat inspiraci při vývinu vlastní aplikace. Každá podkapitola v této části se zabývá jedním typem dokumentace. Všechny dokumentace mají vlastní analýzu, ze kterých jsem nakonec udělal shrnutí a sepsal podklady pro vývin dokumentace.

2.1 Analýza současné webové aplikace

2.1.1 Popis dokumentace firmy

V současné době je firemní dokumentace spíše demonstrační. Na obrázku č. 8 lze vidět úvodní stránku, která obsahuje obecné informace o dokumentaci.



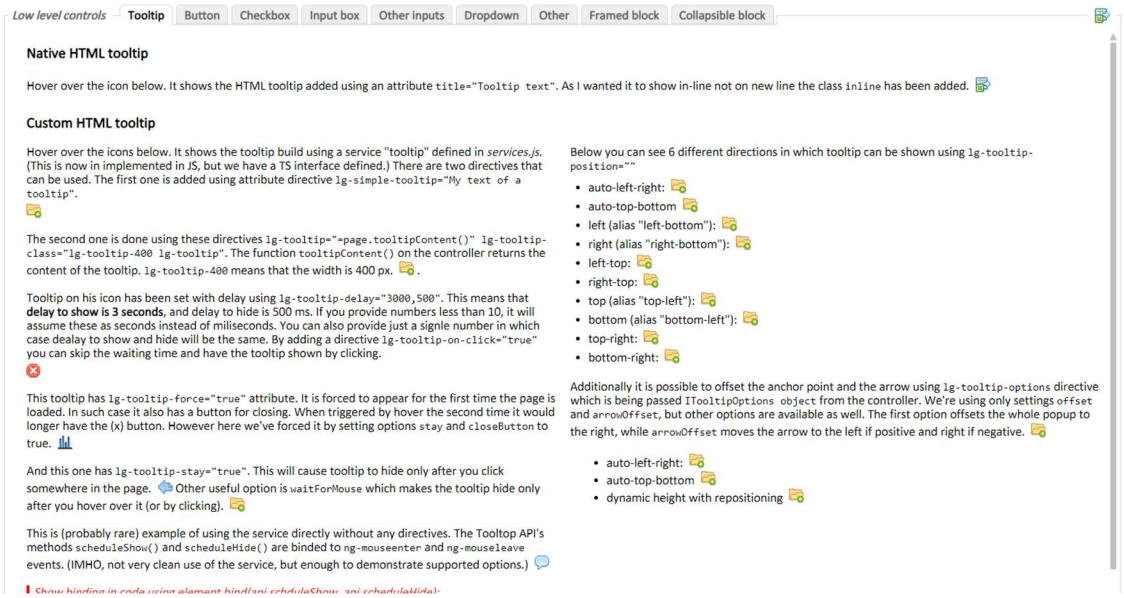
Obrázek č. 8: Dokumentace: úvodní stránka

Stránka je rozdělena do dvou částí. První částí je navigační lišta v horní části, která umožňuje uživateli přecházet na různé komponenty firemního frameworku. Druhá část je pak obsahová. V případě úvodní stránky se jedná o informace o samotném nástroji a doplňující informace pro vývojáře. Další stránky této dokumentace jsou ve stejném stylu, avšak součástí obsahové části jsou již samotné komponenty a jejich popis.

Horní lišta není jediná navigace, kterou v aplikaci lze najít. Na samotných demonstračních stránkách se objevuje další navigace, pomocí které je možné zůstat na stejné stránce a pouze měnit současný pohled na stránku. Tuto navigaci lze vidět na obrázku č. 9.

Obrázek č. 9: Dokumentace – Navigace

Tato navigace je nepřehledná. Pokud se vývojář chce podívat na konkrétní komponentu, musí stránku používat častěji, aby věděl, kde ji může najít. V případě, že k tomuto nástroji přijde nový zaměstnanec, je pro něj velmi náročné se v této dokumentaci vyznat. Aby se uživatel mohl dostat k příkladům rozbalovacích polí, musí přesně vědět, kde má v navigaci hledat.



Obrázek č. 10: Příklad konkrétní dokumentace

Na obrázku č. 10 lze již vidět samotnou prezentaci frameworku. Demonstrace je rozdělená do dvou sloupců a jedna stránka obsahuje větší množství komponent. Stránka zároveň neobsahuje žádný zdrojový kód, který by vývojáři pomohl s vytvářením těchto komponent na své vlastní stránce. Pokud se tedy vývojář chce dozvědět více o

komponentě, musí se podívat do zdrojového kódu celé dokumentace a najít v ní konkrétní prvek, aby zjistil, jakým způsobem jej vytvoří a může s ním manipulovat.

Stránka také neobsahuje žádný popis API konkrétních komponentů. Zmiňuje pouze vybrané prvky, které můžeme specifikovat pro manipulaci chování těchto komponent.

2.1.2 Analýza dokumentace firmy

Dokumentace je spíše zaměřená na prezentaci komponent frameworku, které je možné využít při vývoji aplikací. Jedná se spíše o holistický pohled na framework než na samotné komponenty a neobsahuje zdrojový kód pro vytvoření zmíněných komponent. Pokud chce vývojář nějakou komponentu využít, musí ji nejprve najít, vyzkoušet a až poté ji může využít na vlastní stránce.

V tabulce č. 1 lze vidět SWOT analýzu zpracovanou na současnou dokumentaci firemního frameworku, kterou spolu s dalšími analýzami shrnu na konci této kapitoly.

Tabulka č. 1: SWOT analýza dokumentace LOGEX

-	Dobré	Špatné
Vnitřní	**STRENGTHS**	**WEAKNESSES**
	Základní popis prvků Konkrétní příklady využití, se kterými je možné se setkat Velké množství příkladů u prvků s různou manipulací Nekompletní funkce se udržují	Zaměření je spíše na skupiny než na samostatné prvky Nepřehledná navigace stránky Zdrojový kód není součástí příkladů Není možné nahlédnout do API jednotlivých komponent
Vnější	**OPPORTUNITIES**	**THREATS**
	Využití současných dat a příkladů při vývoji nové dokumentace Nástroj může sloužit jako ukázka pro nové zaměstnance, s čím se mohou setkat	V případě nástupu nových zaměstnanců nebude nástroj dostatečný Udržování dokumentace je náročné a může vést k menší efektivitě zamětnanců

2.2 Analýza dokumentace Angular

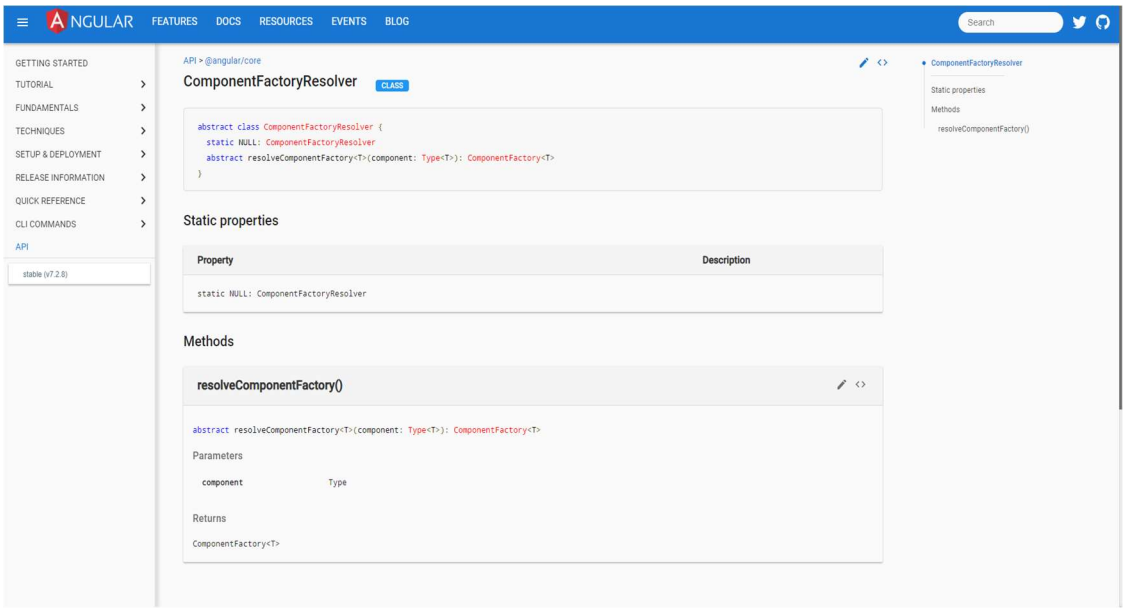
Jelikož v aplikaci využíváme framework Angular a některé komponenty Angular Material, je vhodné se podívat na jejich dokumentaci a zjistit, zda bychom se nemohli inspirovat.

2.2.1 Popis dokumentace frameworku Angular

Z hlediska dokumentace není Angular příliš dobrým příkladem. Nabízí dostatek obecných informací k tomu, jak za využití frameworku psát aplikaci. Mezi tyto informace patří například výpis konvencí, které doporučují vývojářům při vývoji. Jedná se například o pojmenovávání proměnných, souborů a funkcí. Dále Angular nabízí návod na vývoj první aplikace.

Z hlediska dokumentace API však není jasné, zda danou funkci potřebujeme pro naše účely či zda bychom mohli využít něco jiného. Neobsahuje ani bližší informace k tomu, jak konkrétní funkci použít.

Na obrázku č. 11 vidíme například dokumentaci pro *ComponentFactoryResolver*, který slouží k dynamickému generování komponent. Přestože obsahuje využité proměnné a metody, nedává nám žádnou informaci o tom, jak jej použít.



Obrázek č. 11: Dokumentace Angular: ComponentFactoryResolver

2.2.2 Popis dokumentace Angular Material

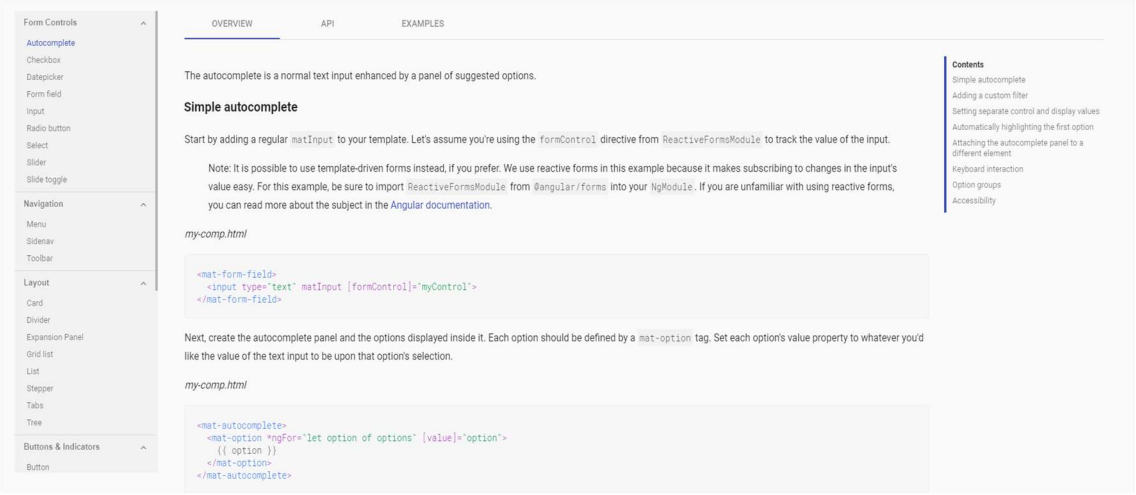
Na rozdíl od samotného frameworku má Angular Material má vynikající dokumentaci. V případě, že chceme využít nějakou komponentu, je velmi snadné se podívat do dokumentace a pochopit, co je potřeba udělat pro správnou funkci komponenty.

Z hlediska komponent se dokumentace skládá ze tří částí. První částí je *Overview*, tedy základní informace o komponentě. Jako jediná záložka obsahuje větší množství textu, který blíže popisuje vytváření komponenty. V této záložce můžeme také najít jednoduché příklady, jejichž součástí je zdrojový kód.

Druhá záložka s názvem *API* obsahuje veškeré veřejné proměnné a funkce, které můžeme nadefinovat či zavolat při používání komponenty. Tato záložka také obsahuje pro každý záznam popis.

Poslední část s názvem *Examples* obsahuje veškeré další příklady využití komponent. I zde je možné se podívat na zdrojový kód.

Na obrázku č. 12 lze vidět dokumentaci jedné z komponent Angular Material, která je otevřená na záložce *Overview* a na příkladu popisuje vytvoření konkrétní komponenty.



Obrázku č. 12: Dokumentace Angular Material: AutoComplete

2.2.3 Analýza dokumentací Angular

Dokumentace samotného frameworku je pro naše účely nedostatečná. Prvky, které jsou na dokumentaci dobré, jsou také součástí dokumentace Angular Material, a většina dokumentace nevyhovuje našim potřebám. Z toho důvodu jsem se rozhodl neprovádět žádnou analýzu na tuto dokumentaci.

Angular Material má velmi kvalitní dokumentaci, ze které je možné čerpat a inspirovat se jí. Rozdělení do jednotlivých částí, vyčerpávající popis samotné komponenty, kterým se můžeme řídit při prvním použití komponenty, a bližší popis API jsou velmi důležité prvky i pro firemní Framework. Na základě této dokumentace jsem zpracoval SWOT analýzu, která je k nahlédnutí v tabulce č. 2.

Tabulka č. 2: SWOT Analýza dokumentace Angular Material

-	Dobré	Špatné
Vnitřní	Strengths	Weaknesses
	Deskriptivní popis API komponent Příklady využití obsahují zdrojový kód. Vyčerpávající popis samotné komponenty Kvalitní navigace na stránce	Dokumentace není plně generovaná ani nevyužívá žádný nástroj, který by usnadnil tvorbu dokumentace.
Vnější	Opportunities	Threats
	Možnost inspirace v této dokumentaci při vytváření vlastního nástroje	Udržování dokumentace může vést ke ztrátě času vývojářů, a tudíž k menší efektivitě.

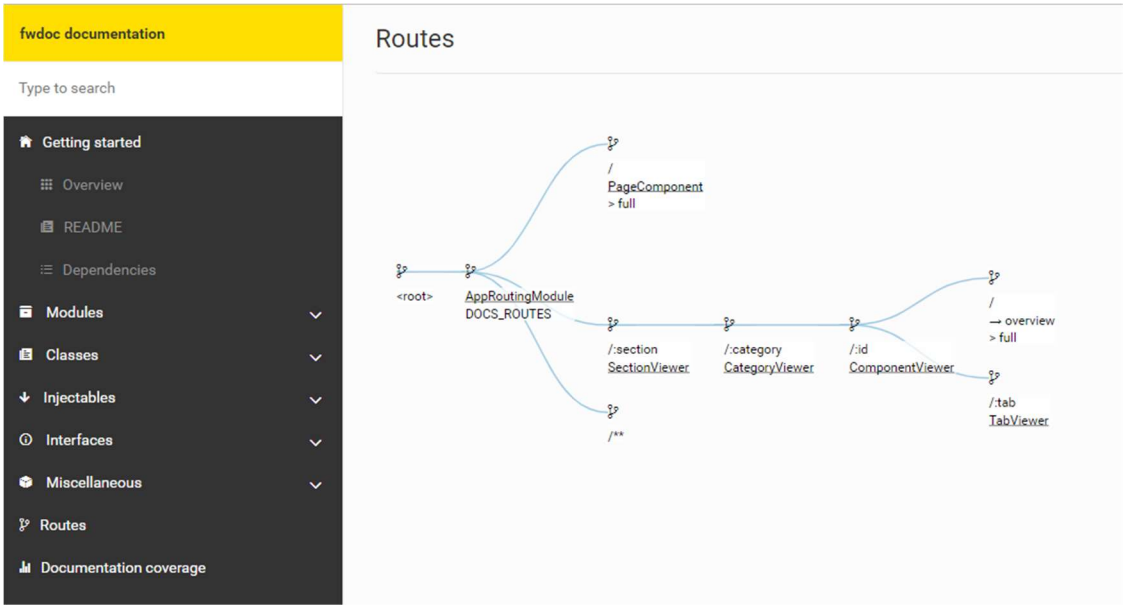
2.3 Analýza nástroje Compodoc

2.3.1 Popis nástroje Compodoc

Jak jsem již zmínil v první kapitole, nástroj Compodoc slouží pro automatickou tvorbu dokumentace na základě obsahu souborů. Tato dokumentace umožňuje rychlou a bezpracnou tvorbu dokumentace. Nicméně tato rychlost a malé úsilí pro vývojáře s sebou

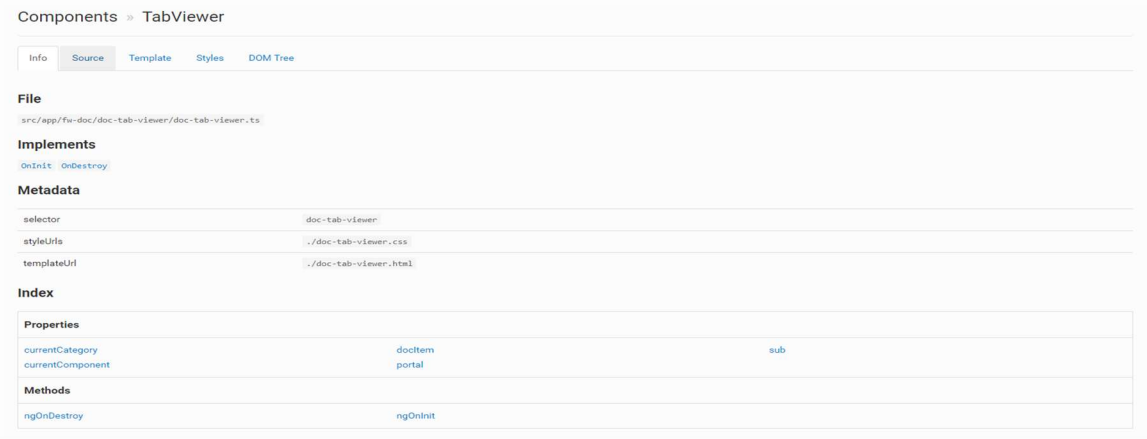
nese jisté problémy. Na rozdíl od ručně vytvořené dokumentace totiž není možné vytvořit příklady pro tyto komponenty, stejně jako není možné psát větší množství textu pro složitější komponenty, u kterých je potřeba více informací.

Tato dokumentace na levé straně obsahuje navigaci, pravou stranu pak zabírá samotný popis. K jedné komponentě se můžeme dostat více způsoby. Skrze záložku *Modules*, které obsahují veškeré komponenty rozdělené dle modulů, anebo skrze *Components*. Dokumentace dále obsahuje všechny *Interfaces*, které byly v aplikaci použity. Nakonec pod záložkou *Miscellaneous* najdeme funkce, které se dokumentačnímu nástroji nepodařilo nikam zařadit, protože nebyly definovány v žádném modulu. Ukázku z dokumentace na záložce *Routes*, která definuje architekturu aplikace, lze vidět na obrázku č. 13.



Obrázek č. 13: Compodoc – navigace a směrovač

Compodoc neumožňuje bližší popis komponent a přidání vlastního obsahu, neboť se jedná o plně automatický nástroj. Přestože není možné přidávat příklady, automaticky generovaná dokumentace může být dobrou referencí při práci s komponenty, se kterými je vývojář obeznámen. Výhodou je také to, že vývojář stráví méně času dokumentací a může se tak více věnovat novým funkcionalitám. Obrázek č. 14 obsahuje pohled na dokumentaci komponenty *TabViewer*, který slouží pro zobrazování různých pohledů na záložky. V dokumentaci můžeme vidět veškeré funkce a proměnné komponenty, DOM strom a také zdrojový kód samotné komponenty včetně HTML a CSS.



Obrázek č. 14: Compodoc – dokumentace jedné komponenty

2.3.2 Analýza nástroje Compodoc

Nástroj Compodoc se v rámci této bakalářské práce nehodí, neboť není možné přidávat vlastní obsah. Aplikace by však měla být alespoň částečně automatická, a z toho důvodu jsem provedl analýzu, kterou lze vidět v tabulka č. 3.

Tabulka č. 3: SWOT Analýza vygenerované dokumentace Compodoc

-	Dobré	Špatné
Vnitřní	Strengths	Weaknesses
	Plně generovaná dokumentace Přehledná navigace na stránce Rozdělení dle funkcionalit, složkové struktury či modularizace Obsahuje přehled pokrytí dokumentace	Není možné upravit prvky prvky dokumentace pro potřeby konkrétních komponent – všechny komponenty jsou zpracovány stejně a není možné přidat příklady či bližší popis vývojářem.
Vnější	Opportunities	Threats
	Je možné využívat tento nástroj společně s jiným dokumentačním nástrojem – nástroj je úplně oddělen od aplikace Nástrojem se lze inspirovat.	Jelikož nejsou k dispozici příklady využití, je možné, že dokumentace nebude dostatečně přínosná pro vývojáře.

2.4 Celkové shrnutí analýzy

V analýzách jsme se podívali na silné a slabé stránky jednotlivých dokumentačních nástrojů. Jelikož analýzy jsou zaměřené na konkrétní nástroje a dokumentace, je obtížné zde najít příležitosti a hrozby. Téměř ve všech analýzách jsme se potýkali s problematikou automatizace. Z toho důvodu by měla aplikace být částečně automatizovaná, avšak měla by obsahovat možnost přidávat další komponenty a tvorbu vlastního obsahu.

Dokumentace také musí být zaměřená na jednotlivé komponenty. Musí být jasné, o jaké komponentě si momentálně čteme, a je také potřeba rozdělení do dvou záložek – obecné informace ke komponentě a výpis API s popisem.

Aplikace by měla obsahovat navigační lištu. Jelikož je vysoká pravděpodobnost, že bude velké množství komponent součástí této lišty, je vhodné přidat vyhledávání v této navigaci, aby si vývojář mohl najít konkrétní komponentu. Je také vhodné, aby součástí aplikace byla možnost filtrování dle rozdělení komponentů – zda se jedná o UI prvky anebo se chceme podívat pouze na služby.

3 VLASTNÍ NÁVRHY ŘEŠENÍ

V této kapitole se zabývám vlastním návrhem řešení. Odkazuji se zde na zdrojový kód, který je k nahlédnutí v přílohách.

3.1 Architektura aplikace

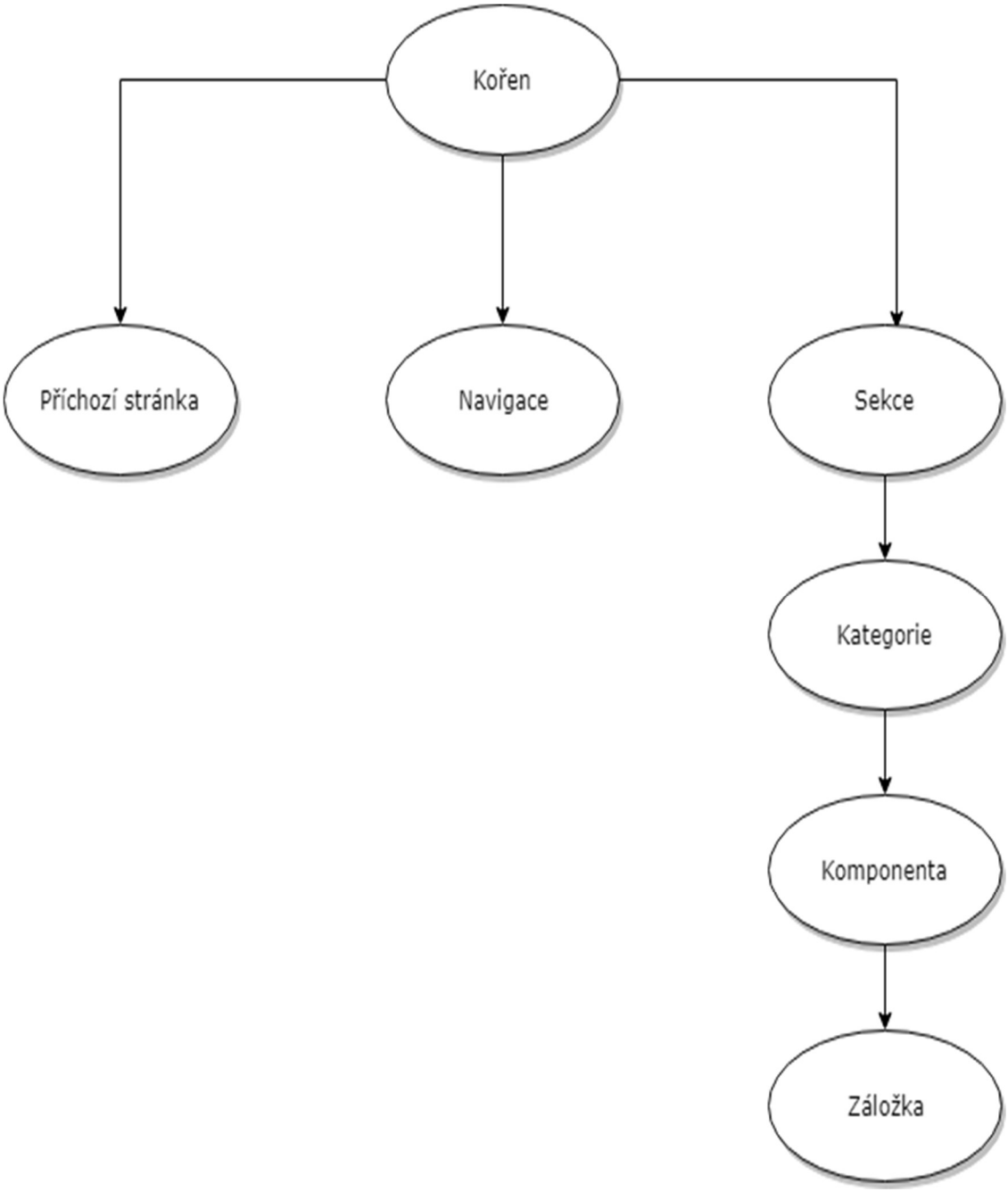
Aplikace je čistě klientská, nevyužívá tedy žádný kód, který provádí operace na serveru. Je možné ji spustit z počítače, který obsahuje zdrojový kód a má nainstalovaný node.js.

Z hlediska architektury se zabývám dvěma typy komponent – *parent* a *child*. *Parent* (rodič) komponenty jsou takové, které pod sebou mají nějaké další komponenty (*child*). Takovou komponentou je například kořen aplikace, *AppComponent*, který je stavebním kamenem celé aplikace a obsahuje všechny další komponenty.

Aplikace se vzhledově skládá ze dvou částí – navigační lišty a samotné dokumentace. Vzhledem k tomu, že komponenty můžeme shlukovat pod různé skupiny, rozhodl jsem se udělat aplikaci ve čtyřech úrovních. První tři úrovně jsou vzhledově identické, mění se pouze nadpisy. Nejnižší úroveň se stará o správu obsahu na základě nadřazených komponent. Tyto úrovně jsou:

- Sekce
- Kategorie
- Komponenta
- Záložka

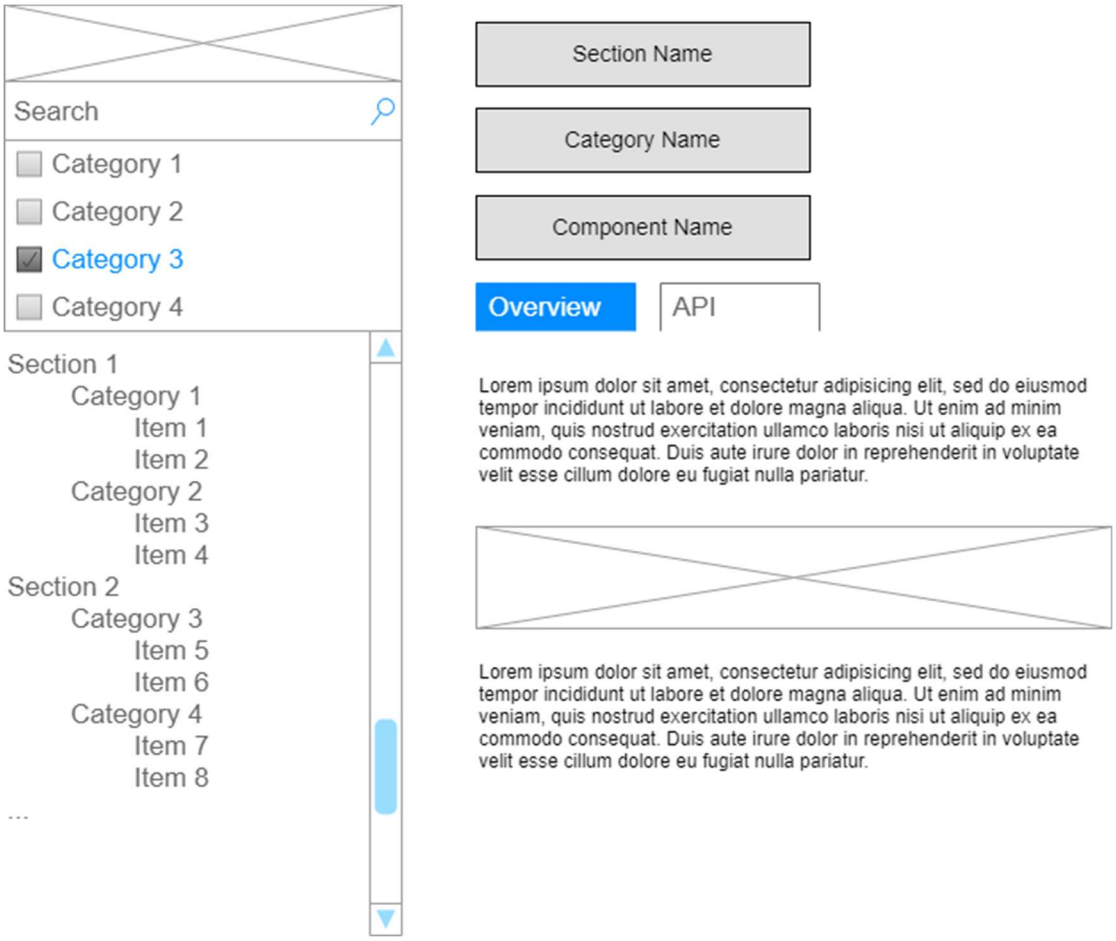
Sekce je nejvyšší úroveň, která je pod kořenem aplikace. Příkladem sekce může být skupina komponent, které se zabývají vzhledem. *Child* komponenta sekce bude *Kategorie*. *Kategorie* může rozdělovat vzhledové funkce podle toho, v jakých částech se používají. Třetí úrovní je již samotná komponenta. Čtvrtou a nejnižší úrovní jsou potom jednotlivé záložky, které popisují komponentu z různých pohledů. Na základě těchto informací jsem udělal prvotní náčrt, který lze vidět na obrázku č. 15.



Obrázek č. 15: Architektura dokumentační aplikace

3.2 Základní návrh vzhledu

Na základě architektury jsem schopný vytvořit wireframe, který bude sloužit jako podklad k vývoji samotné aplikace. Na obrázku č. 16 je možné vidět návrh, kterého se budu snažit docílit v této aplikaci.

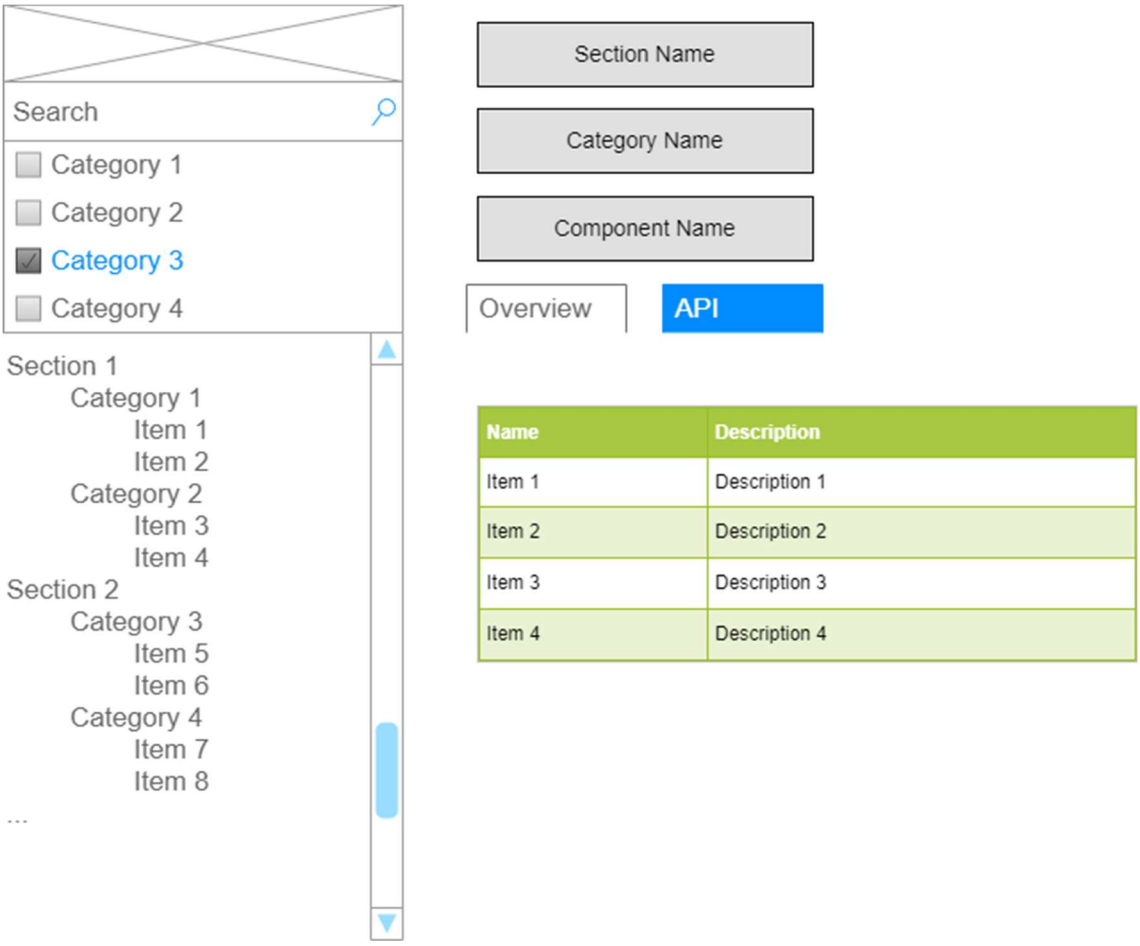


Obrázek č. 16: Wireframe dokumentace (záložka Overview)

Na levé straně je navigační lišta, pomocí které je možné filtrovat různé komponenty, případně vyhledat jednu konkrétní. Vzhledem k tomu, že primárním účelem je využití pro interní framework firmy, v levém horním rohu také počítám s prostorem na logo.

Na pravé straně je pak samotná dokumentace. V horní části můžeme najít název a *Sekce, Kategorie* a *Komponenty*. Pod nimi jsou pak samotné záložky. Pro záložku *Overview* je důležité, aby obsahovala text a místo pro příklady se zdrojovým kódem.

Vzhled stránky pro různé záložky je identický. Jediným rozdílem je obsah. V případě API se totiž jedná pouze o tabulku obsahující názvy parametrů a jejich funkci. Stránku s aktivní záložkou API lze najít na obrázku č. 17.



Obrázek č. 17: Wireframe dokumentace (záložka API)

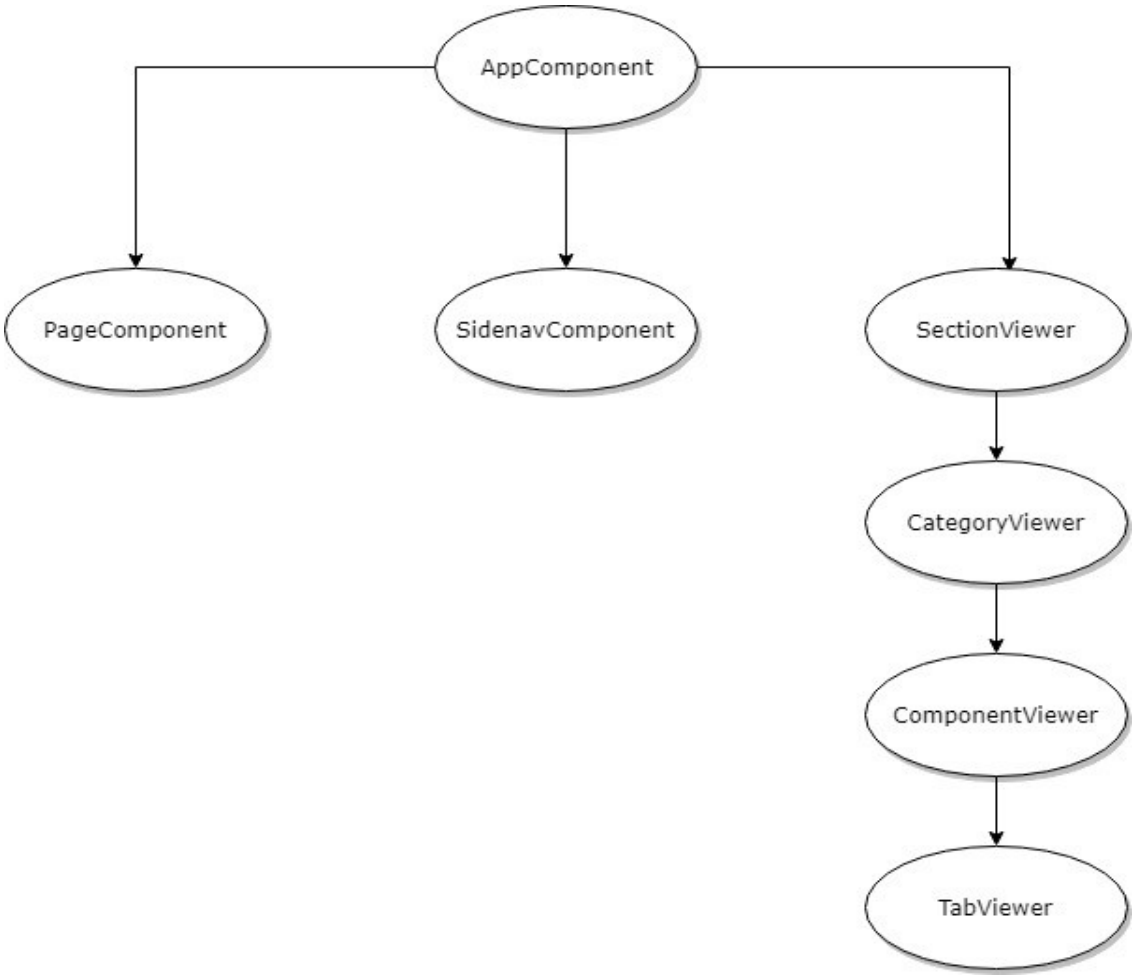
3.3 Směrování aplikace

V závislosti na architektuře a wireframu navrhnu směrování aplikace. V Angularu můžeme využít *RouterModule,* což je modul, který slouží pro směrování. Díky němu lze zachovat některé prvky nezměněné a pouze měnit různé pohledy na stránku. Trasy můžeme mít buď statické, nebo dynamické. V případě statických musíme definovat každou komponentu zvlášť. Jelikož frameworky nabývají velkého množství komponent, je vhodné pro účely této práce využít dynamické směrování. Cesty obsahují parametry, z nichž pro moje účely jsou důležité tři:

- *path* – cesta, která se objeví v URL prohlížeče
- *component* – konkrétní komponent, který budeme zobrazovat na stránce
- *children* – podřazené trasy.

Tyto parametry nám umožňují respektovat výše navrženou hierarchii aplikace. Jelikož však budeme využívat dynamické trasy, musíme vytvořit komponenty, které budou zobrazovat dokumentaci na základě našeho výběru. Trasa obsahuje parametr *component*, který však může obsahovat právě jednu komponentu. Úkolem této komponenty bude právě vyhledat správný *child* komponent, který má stránka obsahovat. Jelikož je mi známa struktura tras aplikace, znám definici URL pro cílovou trasu. Mým úkolem je tedy vytvořit čtyři komponenty, které budou vyhledávat konkrétní stránku – *SectionViewer* pro sekci, *CategoryViewer* pro kategorii, *ComponentViewer* pro komponenty, a nakonec *TabViewer* pro záložky. Na obrázku č. 18 je mírně pozměněný návrh architektury, jsou však na něm představeny konkrétní komponenty.

Další komponenty, které aplikace obsahuje, jsou *SidenavComponent,* který obsahuje navigační lištu, a *PageComponent,* který obsahuje úvodní stránku.



Obrázek č. 18: Architektura dokumentační aplikace v komponentech

3.4 Kořen aplikace

Po specifikaci hierarchie aplikace, směrovače a wireframu mohu psát samotnou aplikaci. Aplikace v Angularu jsou rozdělené do modulů, které pod sebou shromažďují jednotlivé komponenty. Přestože z hlediska komponent je kořenem aplikace *AppComponent*, i ten musí být součástí modulu, a to *AppModule*. Součástí tohoto modulu musí být všechny moduly a komponenty, které aplikace využívá. Samotnou komponentu nemusíme příliš definovat. Nedělá žádné operace, pouze drží aplikaci pohromadě. Zdrojový kód komponenty, modulu a obecných stylů aplikace je možné najít v příloze 1.

3.5 Směrování

V kořenu aplikace také definujeme trasy. Přestože je možné dělat trasy zvlášť pro každý modul, směrovač v této práci díky dynamičnosti není příliš obsáhlý. Samotné komponenty, které využívají směrovač, nejsou ještě vytvořeny. Přesto můžeme díky návrhu z obrázku č. 18 vytvořit definici nyní. Modul obsahující trasy je k nahlédnutí v příloze 2. Směrovač je možné přidat v HTML souboru pod párovým znakem <*router-outlet*>.

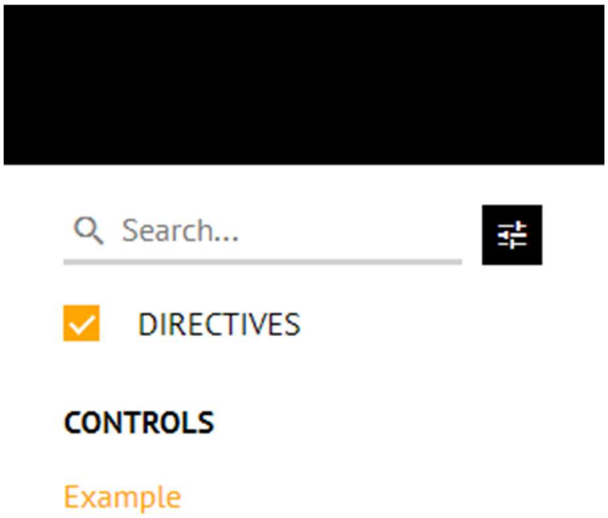
Definice tras obsahuje všechny komponenty kromě kořenového komponentu a navigace. Je to z toho důvodu, že navigace je přítomna na všech stránkách. *PageComponent* a *SectionViewer* jsou tedy definovány v rámci směrovače, zatímco *SidenavComponent* je součástí *AppComponent* na stejné úrovni jako <*router-outlet*>. Struktura tedy není narušena.

Ve směrovači využíváme dynamické směrování. Toho dosáhneme tak, že v definici směrovače pod parametrem *path* přidáme na začátek dvojtečku. Směrovač pak bude veškeré URL parametry přiřazovat konkrétní komponentě. Pod tímto parametrem můžeme také mít dva speciální případy. Prvním je prázdný řetězec, který definuje výchozí komponentu pro cestu. Tento typ je využitý při prvním vstupu na stránku, abychom uživateli zobrazili úvodní stránku, a také při příchodu na konkrétní komponentu, kdy je jako výchozí pohled chceme uživateli zobrazit záložku *Overview*. Druhým případem je pak parametr s dvěma hvězdičkami. Tato možnost je zde pro případ, že nebude možné najít cestu ke komponentě. Směrovač nebude schopný přiřadit trasu a uživatele pošle na

stránku, kterou mu definujeme. V této aplikaci bude uživatel při chybě přesměrován na úvodní stránku.

3.6 Navigační lišta

Navigační lišta obsahuje všechny dostupné sekce, kategorie a komponenty. Uživatelům je dále umožněno vyhledávat konkrétní komponentu.



Obrázek č. 19: Boční navigace

Na obrázku č. 19 je možné vidět boční navigaci. V horní části je možné najít konkrétní komponentu pomocí vyhledávače a filtrace jednotlivých sekcí

Ve spodní části se nachází list samotných komponent. Při kliknutí na nějakou komponentu se pošle příkaz směrovači, aby změnil současný pohled na stránku. Zdrojový kód navigační lišty je možné najít v příloze č. 3.

3.7 Konstanty

Snaha o dynamičnost s sebou nese jisté problémy. Vzhledem ke snaze dosáhnout co nejmenšího množství HTML kódu a dynamickému směrování, musíme v aplikaci uchovávat konstanty, pomocí kterých budeme přiřazovat komponenty při směrování a vytvářet boční navigaci. Aplikace využívá tři konstanty. Všechny tyto konstanty lze nalézt v příloze č. 4.

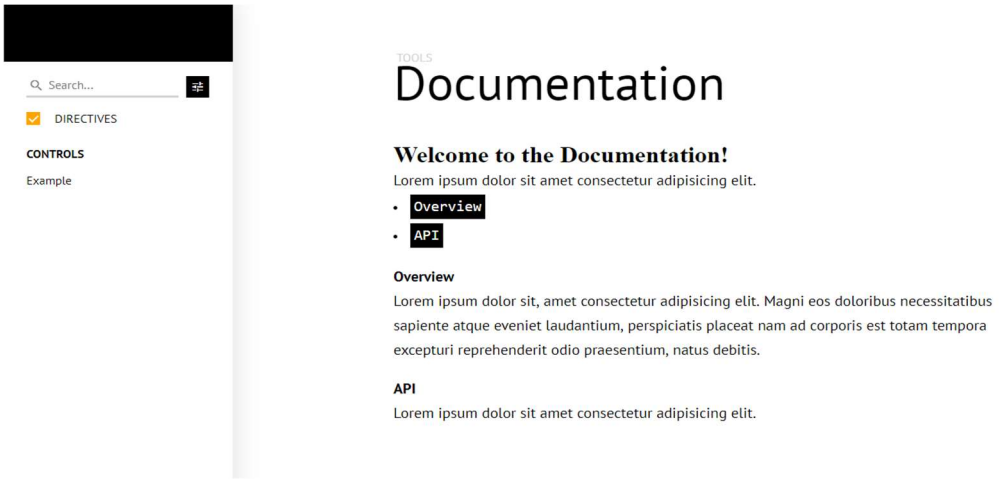
První konstanta *DOCS* udržuje seznam všech komponent a slouží ke generování bočního menu. Tato komponenta má také svoji vlastní službu *DocumentationItems*, která slouží pro extrakci různých dat z této komponenty.

Další konstantou je *ALL_ENTRY_COMPONENTS* definuje komponenty, které jsou generovány dynamicky. Jelikož Angular potřebuje být informovaný o těchto komponentách dříve než za běhu programu, je nutné tyto údaje udržovat.

Poslední konstanta *ALL_COMPONENTS* obsahuje všechny komponenty, které jsou dále využity v komponentě *TabViewer* pro přiřazování pohledů na stránku.

3.8 Úvodní stránka

Po vytvoření boční navigaci se podívám na úvodní stránku. Tato stránka je velmi jednoduchá. Jelikož neobsahuje žádný funkční kód a jedná se pouze o úvod do stránky, je nutné pouze stránku nastylovat a obohatit textem. Výjimkou je služba *ComponentPageTitle,* která slouží k nastavení titulu stránky, který je možné vidět v záložce prohlížeče. Tuto službu budeme potřebovat pro nastavování titulu pro různé komponenty. Úvodní stránku společně s navigací je možné vidět na obrázku č. 20 a zdrojový kód je v příloze č. 5.



Obrázek č. 20: Úvodní stránka

3.9 Tělo aplikace

Po vytvoření úvodní stránky a navigace je možné pracovat na samotné dokumentaci. Ta se skládá ze čtyř komponent. První tři komponenty – *SectionViewer*, *CategoryViewer* a *ComponentViewer* – obsahují pouze směrovač. Přestože neobsahují žádný jiný kód, jsou velmi důležité. Díky tomuto rozdělení je totiž možné poskládat URL ve tvaru *section/category/component/overview*. Na URL se můžeme dále odkazovat a uživatele směrovat na různé pohledy podle této definice. Tyto komponenty je možné najít v příloze č. 6.

3.10 Záložky

Zobrazování samotných zdokumentovaných komponent žije pod komponentou *TabViewer*. V této komponentě využívám konstanty z podkapitoly 7, abych uživateli dodal pohled na jednotlivé komponenty.

HTML soubor komponenty obsahuje pouze záložky a místo, na které má přinášet pohledy, tzv. *Portal*. Funkční část hledá v konstantách konkrétní komponentu na základě URL. *RouterModule* umožňuje pomocí aktivní trasy podívat se na parametry URL a rozdělit je na řetězce.

Řetězec v URL je definovaný v konstantě DOCS a porovnává se s jiným, který je definovaný v konstantě *ALL_COMPONENTS*. V případě, že jsou tyto řetězce identické, na stránku je zaslán pohled na konkrétní komponentu. Z toho důvodu je nutné, aby

identifikační řetězce v konstantě *DOCS* a *ALL_COMPONENTS* byly stejné. Zdrojový kód je k nalezení v příloze č. 7.

Dokumentace je nyní téměř kompletní. Vývojáři mohou vytvářet komponenty a přidat je do konstant tak, aby byly přístupné. Pro každou záložku je tedy nutné vytvořit samostatnou komponentu. Vzhledem k tomu, že je pevně definovaný vzhled záložky API, lze ještě více usnadnit tvorbu dokumentace.

Záložka API obsahuje pouze tabulku s výčtem jednotlivých parametrů a jejich popisem. Vývojáři můžeme dát možnost specifikovat pouze objekt, který bude obsahovat právě tyto parametry. Vývojář tak může namísto přidávání celé nové tabulky a vyplňování jednotlivých buněk přidat pouze objekt a tabulka se vytvoří za něj.

K tomu využiji tzv. *Base Class*. Tato třída bude základem pro jednotlivé komponenty, které budou od této třídy dědit její metody a parametry. Služba *ApiViewerBase* slouží jako základ pro ostatní komponenty, které tvoří dokumentaci. V případě, že vývojář stanoví v dokumentační komponentě nějakou definici, tabulka se automaticky vygeneruje. Pokud tato definice nebude existovat, tabulka se vůbec nevytvoří. Stránka také zobrazuje značku komponenty. Specifikovat tento selektor umožním vývojáři pomocí parametru *_selector*. Příklad záložky API je možné vidět na obrázku č. 21. Zdrojový kód pro službu *ApiViewerBase* a komponentu *ApiTableViewer* lze najít pod přílohou č. 8.

Obrázek č. 21: Záložka API

3.11 Příklady

Poslední funkce, kterou ještě dokumentační nástroj nemá, je možnost zobrazit příklady a jejich zdrojový kód. Zobrazení samotného příkladu je velmi jednoduché. Není zapotřebí žádná speciální komponenta kromě samotného subjektu dokumentace. Jelikož však chceme zobrazit také zdrojový kód, je vhodné přidat samostatnou komponentu, která vzhledově odliší příklad od zbytku stránky a bude starat o zobrazení příkladu.

K tomu slouží modul *ExampleViewerModule.* Tento modul obsahuje dvě komponenty. Komponenta *ExampleViewer* slouží k zobrazení samotného příkladu. *ExampleCodeViewer* se stará o zobrazování zdrojového kódu k příkladům. Jak komponenta vypadá na stránce je možné vidět na obrázku č. 22.

Obrázek č. 22: Komponenta pro zobrazení příkladů

3.11.1 Zobrazení příkladu

Pro zobrazení příkladu tedy využívám komponentu *ExampleViewer*. Tato komponenta obsahuje tři vstupy.

Prvním vstupem je popisek, který bude obsahovat každý příklad.

Druhým vstupem je pak samotný příklad, který se zobrazí uvnitř této komponenty.

Posledním vstupem komponenty je název souboru obsahující kód. V aplikaci kvůli tomu vznikne duplicita. Přestože je možné udělat komponentu, která nebude obsahovat žádnou duplicitu, myslím si, že v tomto případě je vítaná. V opačném případě by totiž byla komponenta složitá. Jedním z cílů práce je, aby bylo snadné pochopit, co se v nástroji děje. Každý vývojář tak bude přesně vědět, na co se komponenta odkazuje, a

bude snadné se v programu vyznat. Tento způsob také usnadňuje vyhledávání podle textu ve složkách. Zdrojový kód této komponenty je možné nalézt v příloze č. 9.

3.11.2 Zobrazení zdrojového kódu

Komponenta pro zobrazení zdrojového kódu je *child* komponentou *ExampleViewer*. Této komponentě je zaslán název souboru, na který se má odkazovat. I tato komponenta se odkazuje na směrovač, není však nutné využití konstant. Ze směrovače zjistíme pouze cestu ke konkrétnímu souboru obsahující zdrojový kód.

Zdrojový kód získávám ze složky *assets*, která slouží k udržování různých zdrojů, na které se můžeme v aplikaci odkazovat. Na tuto složku je možné se také odkazovat pomocí http dotazů.

Struktura složek musí zrcadlit strukturu URL, aby bylo možné se správně odkazovat. V případě, že budeme tedy na komponentě *vzhled/navigace/sipka*, struktura složek musí být *assets/vzhled/navigace/sipka*. V této složce se poté odkážeme na konkrétní soubor, který má stejné jméno, které jsme poslali této komponentě při jejím vytváření.

Obrázek č. 23: Příklad bez zdrojového TypeScript kódu

Komponenta se dívá pouze po souborech se stejným jménem, které jsme jí poslali. Vzhledem k tomu, že příklady bude možné dokumentovat v HTML a TS, je nutné soubory rozlišovat. Rozhodl jsem se pro zakončení souborů, které obsahují zdrojový kód, koncovkou *-html.html* a *-ts.html*. Jelikož se nám vrací jako odpověď http dotazu text, je vhodné mít zdrojový soubor ve formátu *html*. Tento text poté proženeme funkcí knihovny HighlightJS, která nám zabarví příklady, a předáme je na stránku. Na obrázku č. 23 je možné vidět příklad, který neobsahuje žádný TypeScriptový kód. V takovém případě

uživatel dostane hlášku, že příklad kód neobsahuje. Zdrojový kód této komponenty je možné nalézt v příloze č. 10.

3.12 Přidání dokumentace

Nyní, když máme dokumentační nástroj hotový, zbývá pouze přidat nějaký příklad. Pro přidání příkladu je tedy nutné vytvořit novou komponentu. Této komponentě je nutné přidělit nějaké unikátní ID a definovat ji v konstantách *DOCS, ALL_COMPONENTS* a *ALL_ENTRY_COMPONENTS*. Po přidělení těchto ID je možné s touto komponentou pracovat.

Pro *API* vytvoříme specifikujeme pouze parametr *_definition*, který definuje tabulku, a *_selector*, který na stránce řekne uživateli, jaký selektor je potřeba k vytvoření komponenty.

V případě záložky *Overview* se bude jednat spíše o text. Pro přidání příkladu využiji komponentu *ExampleViewer*, kterému pošlu parametry *comment* pro popisek, *fileName* pro název souboru ve složce *assets*, a *template*, který definuje samotný obsah příkladu.

Obrázek č. 24: Příklad dokumentace na záložce Overview

Na obrázku č. 24 je možné vidět příklad záložky Overview, která obsahuje text a *ExampleViewer*. Pro zobrazení zdrojového kódu nakonec vytvořím soubory ve složce *assets* pod stejnou cestou jako URL a mám k dispozici dokumentaci k vybranému subjektu. Zdrojový kód přidané komponenty je k nahlédnutí v příloze č. 11.

4 PŘÍNOS PRÁCE

Na přínos práce se dá dívat ze dvou úhlů.

Prvním je ekonomický přínos práce. Z ekonomického pohledu se nejedná o velký přínos a je velmi složité vyčíslit hodnotu aplikace. Nástroj totiž není výdělečný a nepřinese firmě žádný zisk. Dá se však předpokládat, že firma ušetří peníze na nákladech, protože nebude muset věnovat velké množství času nováčkům. Vzhledem k tomu, že není možné objektivně stanovit čas, který musí vývojáři věnovat nováčkům, a nejsem také seznámen s platy zaměstnanců, ekonomický přínos práce je velmi náročné objektivně posoudit.

Druhým pohledem na přínos je využitelnost. Vzhledem k tomu, že aplikace je ve firmě již zavedená, mohu zpětně říct, že aplikace je nedostatečná. Obsáhlost firemního frameworku dělá aplikaci neudržitelnou. Přestože se jedná o silný nástroj, velké množství komponent je jednoduché a nevyžadují komplexní dokumentaci. Samotný proces dokumentace také zabere čas, kterého vývojáři často nemají dostatek.

ZÁVĚR

Výsledkem bakalářské práce je webová aplikace, která slouží k dokumentaci interních firemních frameworků. Na základě parametrů, které jsem získal při komunikaci s firmou, jsem zanalyzoval současný dokumentační nástroj firmy a porovnal jej se současnými trendy ve vývoji dokumentací. Z různých dokumentačních nástrojů jsem se pokusil vzít ty nejlepší vlastnosti a využít je v této bakalářské práci.

Přestože vývoj aplikace se podařil a došel jsem ke kvalitnímu řešení, po zavedení dokumentačního nástroje ve firmě jsem zpětně zjistil, že řešení je stále nedostatečné. Velké množství komponent je dost jednoduché, nepotřebují tedy komplexní dokumentaci. Řešením by mohla být kombinace plně automatické dokumentace, jako například *compodoc*, a tohoto nástroje. Plně automatická dokumentace může zdokumentovat jednoduché komponenty, které nepotřebují větší množství informací pro práci. Tento nástroj by pak mohl sloužit ke zdokumentování komplexních funkcí a komponent.

Peněžní hodnotu práce je velmi obtížné specifikovat. Aplikace není zisková, avšak může snížit náklady vynaložené na zaučování nováčků. Tyto náklady není možné určit bez interních informací, protože se odvíjí od platu zaměstnanců a času, který vynaloží na zaučování nováčků. Dá se však předpokládat, že nástroj ušetří firmě peníze.

Při zpracování tématu se mi podařilo dosáhnout všech cílů, které jsem stanovil na začátku vývoje. Aplikace je napsána v jazyce TypeScript za využití frameworku Angular a je dost jednoduchá na to, aby do ní mohl kdokoli přidávat dokumentaci. Dokumentace je také dostatečně přehledná i z uživatelského hlediska. Velkou výhodou oproti starší dokumentaci a automatickým nástrojům je také možnost přidání příkladů.

SEZNAM POUŽITÝCH ZDROJŮ

1. NDEGWA, Amos. What is a Web Application?. MaxCDN: StackPath [online]. May 31 2016 [cit. 2018-10-18]. Dostupné z: https://www.maxcdn.com/one/visual-glossary/web-application/
2. Total number of Websites. Internet live stats [online]. [cit. 2018-10-18]. Dostupné z: http://www.internetlivestats.com/total-number-of-websites/
3. Global digital population 2018. Statista [online]. 2018 [cit. 2018-10-18]. Dostupné z: https://www.statista.com/statistics/617136/digital-population-worldwide/
4. Browser. MDN Web Docs [online]. Mozilla, c2005-2018 [cit. 2018-10-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Browser
5. Internet Explorer. Microsoft [online]. c2018 [cit. 2018-10-19]. Dostupné z: https://www.microsoft.com/en-us/download/internet-explorer.aspx
6. Google Chrome Web Browser. Google [online]. [cit. 2018-10-19]. Dostupné z: https://www.google.com/chrome/
7. Download Firefox: Free Web Browser. Mozilla: Internet for people, not profit [online]. c1998-2018 [cit. 2018-10-19]. Dostupné z: https://www.mozilla.org/en-US/firefox/new/
8. Safari: Apple. Apple [online]. Apple, c2018 [cit. 2018-10-19]. Dostupné z: https://www.apple.com/lae/safari/
9. ECMAScript 6 compatibility table. GitHub: The world's leading software development platform [online]. GitHub, 2007 [cit. 2018-10-19]. Dostupné z: http://kangax.github.io/compat-table/es6/
10. Browser statistics. W3Schools Online Web Tutorials [online]. Refsnes Data, c1999-2018 [cit. 2018-10-19]. Dostupné z: https://www.w3schools.com/browsers/
11. Browser market share. Net Applications [online]. c2017 [cit. 2018-10-19]. Dostupné z: https://netmarketshare.com/browser-market-share.aspx?
12. StatCounter Global Stats: Browser, OS, Search Engine including Mobile Usage Share. StatCounter: Web Analytics Made Easy [online]. c1999-2017 [cit. 2018-10-19]. Dostupné z: http://gs.statcounter.com/
13. Document Object Model. MDN Web Docs [online]. Mozilla, c2005-2019 [cit. 2018-10-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
14. Render Functions & JSX: Vue.js. Vue.js [online]. Evan You, c2014-2019 [cit. 2018-10-18]. Dostupné z: Virtual DOM: https://vuejs.org/v2/guide/render-function.html#The-Virtual-DOM
15. Search. GitHub: The world's leading software development platform [online]. GitHub, 2007 [cit. 2018-10-19]. Dostupné z: https://github.com/search?o=desc&p=1&q=stars%3A%3E1&s=stars&type=Repositories
16. Introduction to HTML. W3Schools Online Web Tutorials [online]. Refsnes Data, c1999-2018 [cit. 2018-10-19]. Dostupné z: https://www.w3schools.com/html/html_intro.asp
17. D3.js: Data-Driven Documents [online]. c2017 [cit. 2018-10-19]. Dostupné z: https://d3js.org/

18. Document Structure: SVG 1.1 (Second Edition). World Wide Web Consortium [online]. W3C, c2018 [cit. 2018-10-19]. Dostupné z: https://www.w3.org/TR/SVG11/struct.html#SVGElement
19. HTML Canvas. W3Schools Online Web Tutorials [online]. Refsnes Data, c1999-2018 [cit. 2018-10-19]. Dostupné z: https://www.w3schools.com/graphics/canvas_intro.asp
20. WebGL: 2D and 3D graphics for the web. MDN Web Docs [online]. Mozilla, c2005-2018 [cit. 2018-10-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
21. Manual: Getting started with WebGL Development. Unity [online]. Unity Technologies, c2018 [cit. 2018-10-19]. Dostupné z: https://docs.unity3d.com/Manual/webgl-gettingstarted.html
22. CSS Introduction. W3Schools Online Web Tutorials [online]. c1999-2018 [cit. 2018-10-19]. Dostupné z: https://www.w3schools.com/css/css_intro.asp
23. CSS Animations. W3Schools Online Web Tutorials [online]. Refsnes Data, c1999-2018 [cit. 2018-10-19]. Dostupné z: https://www.w3schools.com/css/css3_animations.asp
24. CSS Media Queries. W3Schools Online Web Tutorials [online]. Refsnes Data, c1999-2018 [cit. 2018-10-19]. Dostupné z: https://www.w3schools.com/css/css3_mediaqueries.asp
25. Sass: Syntactically Awesome Style Sheets [online]. c2006-2018 [cit. 2018-10-19]. Dostupné z: https://sass-lang.com/
26. Sass: Documentation. Sass: Syntactically Awesome Style Sheets [online]. c2006-2018 [cit. 2018-10-19]. Dostupné z: https://sass-lang.com/documentation
27. JavaScript Tutorial. W3Schools Online Web Tutorials [online]. Refsnes Data, c1999-2018 [cit. 2018-10-19]. Dostupné z: https://www.w3schools.com/js/
28. Is JavaScript Single-Threaded? Redgate Hub [online]. Red Gate Software, c1999-2018 [cit. 2018-10-19]. Dostupné z: https://www.red-gate.com/simple-talk/dotnet/asp-net/javascript-single-threaded/
29. About JavaScript. MDN Web Docs [online]. Mozilla, c2005-2018 [cit. 2018-10-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript
30. Node.js [online]. Node.js Foundation [cit. 2018-10-19]. Dostupné z: https://nodejs.org/en/
31. V8. GitHub: The world's leading software development platform [online]. GitHub, 2007 [cit. 2018-10-19]. Dostupné z: https://github.com/v8
32. Angular: Getting Started. Angular.io [online]. Google, c2010-2018 [cit. 2018-10-19]. Dostupné z: https://angular.io/guide/quickstart
33. D3 Mouse Events: bl.ocks.org. Bl.ocks.org: Popular Blocks [online]. 2010 [cit. 2018-10-19]. Dostupné z: http://bl.ocks.org/WilliamQLiu/76ae20060e19bf42d774
34. TensorFlow.js [online]. TensorFlow [cit. 2018-10-19]. Dostupné z: https://www.tensorflow.org/js/
35. Electron [online]. [cit. 2018-10-18]. Dostupné z: https://electronjs.org/

36. TypeScript: JavaScript that scales [online]. Redmond: Microsoft, c2012-2018 [cit. 2018-12-30]. Dostupné z: https://www.typescriptlang.org/
37. TypeScript: TypeScript in 5 minutes. TypeScript: JavaScript that scales [online]. Redmond: Microsoft, c2012-2018, c2012-2018 [cit. 2018-12-30]. Dostupné z: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html
38. Babel: The compiler for next generation JavaScript [online]. [cit. 2018-12-30]. Dostupné z: https://babeljs.io/
39. MDN: Browser support for JavaScript APIs. MDN Web Docs [online]. Mozilla, c2005-2018, 27 Jun 2018 [cit. 2018-12-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Browser_support_for_JavaScript_APIs
40. BARBIER, Alexandre. Polyfills and Transpilers: one code for every browser. Medium: A place to read and write big ideas and important stories [online]. Oct 29 2015 [cit. 2018-12-30]. Dostupné z: https://medium.com/@alexbrbr/polyfills-and-transpilers-one-code-for-every-browser-abaa85145c9c
41. ROBERTS, Philip. JSConf EU: What the heck is event loop anyway?. YouTube [online video]. 9. 10. 2014 [cit. 2018-12-30]. Dostupné z: https://www.youtube.com/watch?v=8aGhZQkoFbQ
42. ROBERTS, Philip. Loupe [online]. 9 Mar 2014 [cit. 2018-12-30]. Dostupné z: http://latentflip.com/loupe/
43. RAVEENDRAN, Anoop. JavaScript Event Loop Explained. Medium: a place to read and write big ideas and important stories [online]. 27 Dec 2017 [cit. 2018-12-30]. Dostupné z: https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4
44. ARCHIBALD, Jake. JSConf Asia 2018: In the loop. YouTube [online video]. 9. 2. 2018 [cit. 2018-12-30]. Dostupné z: https://www.youtube.com/watch?v=cCOL7MC4Pl0
45. MDN: Introduction to web APIs. MDN Web Docs [online]. Mozilla, c2005-2018, Aug 29, 2018 [cit. 2018-12-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction
46. MDN: Fetch API. MDN Web Docs [online]. Mozilla, c2005-2018, 23 Oct 2018 [cit. 2018-12-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
47. Twitter Developer Platform [online]. Twitter, c2018 [cit. 2018-12-30]. Dostupné z: https://developer.twitter.com/
48. MDN: Window. MDN Web Docs [online]. Mozilla, c2005-2018, 20 Nov 2018 [cit. 2018-12-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Window
49. JakeArchibald.com: Tasks, microtasks, queues and schedules. JakeArchibald.com: Blog [online]. 17 Aug 2015 [cit. 2018-12-30]. Dostupné z: https://jakearchibald.com/2015/tasks-microtasks-queues-and-schedules/
50. GTmetrix: Website Speed and Performance Optimization [online]. GT.net, c2018 [cit. 2018-12-30]. Dostupné z: https://gtmetrix.com/
51. PageSpeed Insights [online]. [cit. 2018-12-30]. Dostupné z: https://developers.google.com/speed/pagespeed/insights/

52. Google PageSpeed Rules for Dummies: What Do They Mean?. Webris: The Digital Marketing Agency [online]. Webris, c2018, c2018 [cit. 2018-12-30]. Dostupné z: https://webris.org/google-pagespeed-rules-for-dummies-what-do-they-mean/
53. JavaScript Frameworks by the Numbers: Winter 2018. JavaScript Report [online]. 5 Feb 2018 [cit. 2018-12-30]. Dostupné z: https://jsreport.io/javascript-frameworks-by-the-numbers-winter-2018/
54. AngularJS API: API Reference. Angular JS: Superheroic JavaScript MVW Framework [online]. Google, c2010-2018, c2010-2018 [cit. 2018-12-30]. Dostupné z: https://docs.angularjs.org/api
55. Contributing to AngularJS. GitHub: The world's leading software development platform [online]. GitHub, 2007, 11 Nov 2017 [cit. 2018-12-30]. Dostupné z: https://github.com/angular/angular.js/blob/master/CONTRIBUTING.md
56. AngularJS: Angular, version 2. Angular JS [online]. Google, c2010-2018, 14 Sep 2016 [cit. 2018-12-30]. Dostupné z: https://blog.angularjs.org/2016/09/angular2-final.html
57. Angular Docs [online]. Google, c2010-2018 [cit. 2018-12-30]. Dostupné z: https://angular.io/
58. Dynamic Component Loader. Angular [online]. Google, c2010-2018 [cit. 2019-01-24]. Dostupné z: https://angular.io/guide/dynamic-component-loader
59. Angular Material [online]. Google, c2010-2018 [cit. 2019-02-20]. Dostupné z: https://material.angular.io/
60. Component infrastructure and Material Design components for Angular. GitHub [online]. San Francisco: GitHub, 2007 [cit. 2019-02-20]. Dostupné z: https://github.com/angular/material2
61. React: A declarative, efficient, and flexible JavaScript Library for building user interface. GitHub: The world's leading software development platform [online]. GitHub, 2007 [cit. 2018-12-30]. Dostupné z: https://github.com/facebook/react/
62. React: Introducing JSX. React: A JavaScript library for building user interfaces [online]. Facebook, c2018, c2018 [cit. 2018-12-30]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html
63. React: ReactDOM. React: A JavaScript library for building user interfaces [online]. Facebook, c2018, c2018 [cit. 2018-12-30]. Dostupné z: https://reactjs.org/docs/react-dom.html
64. Vue.js: A progressive, incrementally-adoptable JavaScript framework for building UI on the web. GitHub [online]. GitHub, 2007 [cit. 2018-12-30]. Dostupné z: https://github.com/vuejs/vue
65. YOU, Evan. Vue.js [online]. c2014-2018 [cit. 2018-12-30]. Dostupné z: https://vuejs.org/
66. *Material Design* [online]. Google [cit. 2019-04-01]. Dostupné z: https://material.io/
67. Highlight.js [online]. Ivan Sagalaev, 2006 [cit. 2019-02-20]. Dostupné z: https://highlightjs.org/
68. JavaScript syntax highlighter. GitHub [online]. San Francisco: GitHub, 2007 [cit. 2019-02-20]. Dostupné z: https://github.com/highlightjs/highlight.js

69. What is wireframing: Experience UX. Experience UX: We're UX agency specialising in UX research [online]. Bournemouth: Experience UX, c2018 [cit. 2019-02-20]. Dostupné z: https://www.experienceux.co.uk/faqs/what-is-wireframing/
70. OGLOBLINSKY, Vincent. Compodoc: The missing documentation tool for your Angular application. *GitHub: a place to read and write big ideas and important stories* [online]. GitHub, c2019, c2016-2019 [cit. 2019-03-11]. Dostupné z: https://github.com/compodoc/compodoc
71. SWOT Analysis. Investopedia: Sharper Insight, Smarter Investing [online]. Dotdash, 23 May 2018 [cit. 2018-12-30]. Dostupné z: https://www.investopedia.com/terms/s/swot.asp
72. SWOT Analysis: Strengths, Weaknesses, Opportunities, Threats. Management Training and Leadership [online]. London: Mind Tools, c1996-2018 [cit. 2018-12-30]. Dostupné z: https://www.mindtools.com/pages/article/newTMC_05.htm

SEZNAM POUŽITÝCH ZKRATEK A SYMBOLŮ

DOM	-	Document Object Model
HTML	-	Hypertext Markup Language
WebGL	-	Web Graphics Library
CSS	-	Cascading Style Sheets
SCSS	-	Sassy Cascading Style Sheets
JS	-	JavaScript
ES	-	ECMAScript
API	-	Application Programming Interface
CLI	-	Command Line Interface
JSX	-	JavaScript eXtension
SWOT	-	Strength, Weaknesses, Opportunities and Threats
URL	-	Uniform Resource Locator

SEZNAM TABULEK

SEZNAM OBRÁZKŮ

SEZNAM GRAFŮ

SEZNAM PŘÍLOH

Příloha č. 1: Kořen aplikace

app.component.ts

```
import { Component } from '@angular/core';

@Component({
    selector: 'app-root',
    template: `
        <doc-sidenav></doc-sidenav>
        <div class="content">
            <router-outlet></router-outlet>
        </div>`,
})
export class AppComponent { }
```

app.module.ts

```
@NgModule( {
    declarations: [
        AppComponent,
    ],
    imports: [
        FrameworkDocumentationModule,
        DocumentationComponentsModule,
        SharedModule,
        AppRoutingModule,
        HttpClientModule
    ],
    providers: [
      HttpClient
    ],
    bootstrap: [AppComponent]
} )
export class AppModule { }
```

styles.scss

```
@import "@angular/material/prebuilt-themes/deeppurple-amber.css";
@import "~@angular/cdk/overlay-prebuilt.css";
@import "~assets/lib/hljs/styles/hljs.css";

* {
    margin: 0;
    padding: 0;
}

.content {
    box-sizing: border-box;
    padding-left: 210px;
```

```
    padding-right: 210px;
    margin-left: 300px;
    padding-bottom: 200px;
}

.tab-content,
.landing-page {
    width: calc(100% * 8 / 12);
}

.landing-page {
    margin-top: 40px;
}

.category-title,
.landing-page-title {
    display: block;
    font-family: "PT Sans", sans-serif;
    font-size: 16px;
    line-height: 16px;
    color: lightgray;
    text-transform: uppercase;
    position: absolute;
    margin-left: 4px;
    user-select: none;
    -ms-user-select: none;
    -moz-user-select: none;
    -webkit-user-select: none;
}

.component-title,
.landing-page-tool-title {
    display: block;
    margin-top: 70px;
    font-family: "PT Sans", sans-serif;
    font-size: 66px;
    line-height: 85px;
    color: black;
    user-select: none;
    -ms-user-select: none;
    -moz-user-select: none;
    -webkit-user-select: none;
}

.row {
    clear: both;
}
```

```
h2,
p {
    font-family: "PT Sans", sans-serif;
    font-size: 20px;
    line-height: 34px;
    color: black;
}

ul {
    list-style-position: inside;
}

.keyword {
    display: inline-block;
    font-family: Consolas, serif;
    font-size: 20px;
    line-height: 34px;
    color: white;
    padding-left: 5px;
    padding-right: 5px;
    background-color: black;
    margin-top: 3px;
}

ul li span.keyword {
    margin-bottom: 3px;
}

.text-block {
    margin-bottom: 20px;
}

.table {
    border: 2px solid lightgray;
    margin-bottom: 40px;
    margin-top: 10px;
    border-collapse: collapse;
    margin-left: -20px;
}

.table th,
.table td {
    font-size: 18px;
    line-height: 34px;
    border: 2px solid lightgray;
    padding-left: 20px;
```

```
    padding-right: 20px;
}

.table th {
    font-family: "PT Sans", sans-serif;
    color: lightgray;
    text-align: left;
}

.table td:first-child {
    font-family: Consolas, serif;
    color: black;
}

.table td:last-child {
    font-family: "PT Sans", sans-serif;
    color: black;
}
```

Příloha č. 2: AppRoutingModule

app-routing-module.ts

```
export const DOCS_ROUTES: Routes = [
    { path: '', component: PageComponent, pathMatch: 'full' },
    {
        path: ':section',
        component: SectionViewer,
        children: [
            {
                path: ':category',
                component: CategoryViewer,
                children: [
                    {
                        path: ':id',
                        component: ComponentViewer,
                        children: [
                            { path: '', redirectTo: 'overview',
pathMatch: 'full' },
                            { path: ':tab', component: TabViewer }
                        ],
                    },
                ]
            },
        ],
    },
    { path: '**', redirectTo: '' }
];


@NgModule( {
    imports: [
        RouterModule.forRoot( DOCS_ROUTES, {
            useHash: true,
            enableTracing: false,
        } )
    ],
    exports: [
        RouterModule
    ],
    providers: [
        { provide: APP_BASE_HREF, useValue: "!"}
    ]
} )

export class AppRoutingModule { }
```

Příloha č. 3: SidenavComponent

doc-sidenav-component.html

```
<div class="sidenav-container">
    <div class="sidenav-header">
        <a>
            <div class="logo"> </div>
        </a>
    </div>
    <div class="sidenav-filter-wrapper">
        <div class="row">
            <div class="filter">
                <div class="search">
                    <input type="text" placeholder="Search..."
[(ngModel)]="_fullTextSearchInput"
                        (keyup)="changeList()">
                    <i class="material-icons mat-search"
(click)="changeList()"> search </i>
                </div>
                <i class="material-icons mat-tune"
(click)="_toggleSectionFilter()"
                    [class.mat-tune-active]="_visibleSectionFilter">
                    tune </i>
            </div>
        </div>
        <div class="row">
            <div class="section-filter" *ngIf="_visibleSectionFilter">
                <div *ngFor="let section of _sections; index as i;"
class="section-filter-row">
                    <label for="{{section}}" (click)="_toggleCheckbox(i)"
class="checkbox-container">
                        <input type="checkbox"
[checked]="_shownSections[i]" name="{{section}}" class="checked">
                        <span class="checkmark"></span>
                        <span class="checkbox-label">{{section |
uppercase }}</span>
                    </label>
                </div>
            </div>
        </div>
    </div>
    <div [ngClass]="{'list-container-filter-open':_visibleSectionFilter,
    'list-container-filter-closed':!_visibleSectionFilter}">
        <div class="list">
            <div class="sidenav-list-wrapper">
                <div class="row">
```

```
                    <div *ngFor="let category of
((_categoriesList|async)); index as i; trackBy: _trackByFn">
                        <div class="category-name"
(click)="_toggleCategory(i)"
                            [class.category-name-
active]="_shownCategories[i]">
                            <b>{{category.name}}</b>
                        </div>
                        <div class="components"
*ngIf="_shownCategories[i]">
                            <div
                                *ngFor="let component of
(_componentsList|async) as filteredItems; trackBy: _trackByFn">
                                <div [routerLink]="[component.sectionId,
component.categoryId, component.id]"
                                    routerLinkActive="active-component"
class="component"

*ngIf="component.categoryId==category.id"> {{component.name}} </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

doc-sidenav-component.ts

```
@Component( {
    selector: 'doc-sidenav',
    styleUrls: ['./doc-sidenav.component.scss'],
    templateUrl: './doc-sidenav.component.html'
} )
export class SidenavComponent {
    _sections: string[];
    _componentsList: BehaviorSubject<DocItem[]> = new BehaviorSubject( [] 
);
    _categoriesList: BehaviorSubject<DocCategory[]> = new
BehaviorSubject( [] );

    _shownSections: boolean[] = [];
    _shownCategories: boolean[] = [];
    _fullTextSearchInput: string;
```

```
    _visibleSectionFilter: boolean;

    constructor(
        private _docItems: DocumentationItems,
    ) {
        this._fullTextSearchInput = '';
        this._sections = Object.keys( this._docItems.getDocs() );

        this._sections.forEach( () => {
            this._shownSections.push( true );
        } );

        this._docItems.getAllCategories().forEach( () => {
            this._shownCategories.push( true );
        } );
        this._visibleSectionFilter = true;

        this.changeList();
    }

    public _toggleSectionFilter(): void {
        this._visibleSectionFilter = !this._visibleSectionFilter;
    }

    public _toggleCategory( id: number ): void {
        this._shownCategories[id] = !this._shownCategories[id];
    }

    public _trackByFn( index, entry: DocItem ): string {
        return '' + index + entry.name;
    }

    public _toggleCheckbox( id: number ): void {
        this._shownSections[id] = !this._shownSections[id];
        this.changeList();
    }

    changeList() {
        if ( this._shownSections.every( x => !x ) ) {
            this._emitUpdatedList( [], [] );
            return;
        }
        let componentsForOutput: DocItem[];
        let categoriesForOutput: DocCategory[];
        if ( this._shownSections.every( x => x ) ) {
            componentsForOutput = this._getComponentsForOutput();
```

```
            categoriesForOutput = this._getCategoriesForOutput(
componentsForOutput );

            this._emitUpdatedList( categoriesForOutput,
componentsForOutput );
            return;
        }

        const selectedSections: string[] = this._getSelectedSections(
this._sections, this._shownSections );

        componentsForOutput = this._getComponentsForOutput(
selectedSections );
        categoriesForOutput = this._getCategoriesForOutput(
componentsForOutput );

        this._emitUpdatedList( categoriesForOutput, componentsForOutput
);
    }

    private _getCategoriesForOutput( components: DocItem[] ):
DocCategory[] {
        const categories = this._docItems.getAllCategories();
        const filteredCategories: DocCategory[] =
this._getFilteredCategories( categories, components );
        const sortedCategories: DocCategory[] =
this._getSortedItems<DocCategory>( filteredCategories );
        return sortedCategories;
    }

    private _getComponentsForOutput( sections?: string[] ): DocItem[] {
        const allItems = this._docItems.getAllItems();
        let filteredComponents: DocItem[];
        let sortedComponents: DocItem[];
        if ( sections && sections.length ) {
            const componentListForSearch: DocItem[] =
this._getResultList( sections );
            filteredComponents = this._getItemsByFullTextSearch(
componentListForSearch );
            sortedComponents = this._getSortedItems<DocItem>(
filteredComponents );
            return sortedComponents;
        }

        filteredComponents = this._getItemsByFullTextSearch( allItems );
        sortedComponents = this._getSortedItems<DocItem>(
filteredComponents );
```

```
        return sortedComponents;
    }

    private _getSelectedSections( allSections: string[], showSections:
boolean[] ): string[] {
        return allSections.filter( ( section, index ) =>
showSections[index] );
    }

    private _getResultList( selectedSections: string[] ): DocItem[] {
        return selectedSections.reduce( ( result: DocItem[], current:
string ) => {
            return [...result, ...this._docItems.getItemsBySection(
current )];
        }, [] );
    }

    private _getItemsByFullTextSearch( items: DocItem[] ): DocItem[] {
        if ( this._fullTextSearchInput === '' ) {
            return items;
        }
        return items.filter( item => item.name.toLowerCase().includes(
this._fullTextSearchInput.toLowerCase() ) );
    }

    private _getFilteredCategories( categories: DocCategory[],
components: DocItem[] ): DocCategory[] {
        return categories.filter( category => {
            return components.find( component => component.categoryId ===
category.id );
        } );
    }

    private _getSortedItems<TItems extends DocCategory | DocItem>( items:
TItems[] ): TItems[] {
        return items.sort( ( a, b ) => {
            if ( a.id < b.id ) {
                return -1;
            }
            if ( a.id > b.id ) {
                return 1;
            }
            return 0;
        } );
    }
```

```
    private _emitUpdatedList( categories: DocCategory[], components:
DocItem[] ): void {
        this._categoriesList.next( categories );
        this._componentsList.next( components );
    }
}
```

doc-sidenav-component.scss

```scss
.sidenav-container {
    position: fixed;
    height: 100%;
    width: 300px;
    background-color: white;
    box-shadow: 4px 0px 30px rgba(0, 0, 0, 0.1);
    top: 0;
    left: 0;
    overflow: hidden;

    .row {
        clear: both;
    }

    .sidenav-header {
        box-sizing: border-box;
        width: 100%;
        height: 84px;
        background-color: black;
        padding: 30px 26px;
    }

    .sidenav-filter-wrapper,
    .sidenav-list-wrapper {
        width: calc(100% * 0.8);
        padding-left: calc(100% * 0.1);
        padding-right: calc(100% * 0.1);
        padding-bottom: 20px;
    }

    .sidenav-filter-wrapper {
        padding-bottom: 30px;

        .filter {
            margin-top: 20px;
            cursor: pointer;
        }
```

```
.search {
    float: left;
    padding-right: 10px;
}

.mat-search {
    box-sizing: border-box;
    float: left;
    font-size: 20px;
    padding: 3px 3px 4px 3px;
    color: gray;
    overflow: visible;
    transition: all 0.25s;
    -o-transition: all 0.25s;
    -moz-transition: all 0.25s;
    -webkit-transition: all 0.25s;
    user-select: none;
    -ms-user-select: none;
    -moz-user-select: none;
    -webkit-user-select: none;
}

.search input[type="text"] {
    width: 170px;
    border: none;
    color: gray;
    padding-top: 2px;
    padding-bottom: 2px;
    margin-left: 3px;
    font-family: "PT Sans", sans-serif;
    font-size: 16px;
    cursor: pointer;
    transition: all 0.25s;
    -o-transition: all 0.25s;
    -moz-transition: all 0.25s;
    -webkit-transition: all 0.25s;
}

.mat-tune {
    padding: 6px;
    font-size: 18px;
    color: gray;
    background-color: lightgray;
    cursor: pointer;
    transition: all 0.25s;
    -o-transition: all 0.25s;
```

```
        -moz-transition: all 0.25s;
        -webkit-transition: all 0.25s;
        user-select: none;
        -ms-user-select: none;
        -moz-user-select: none;
        -webkit-user-select: none;
    }

    .mat-tune:hover,
    .mat-tune-active {
        background-color: black;
        color: white;
    }

    .search::after {
        position: relative;
        display: block;
        left: 0;
        bottom: 0px;
        content: "";
        height: 3px;
        width: 100%;
        background-color: lightgray;
        clear: both;
        transition: all 0.25s;
        -o-transition: all 0.25s;
        -moz-transition: all 0.25s;
        -webkit-transition: all 0.25s;
    }

    .search:hover::after {
        background-color: #cccccc;
        transition: all 0.25s;
        -o-transition: all 0.25s;
        -moz-transition: all 0.25s;
        -webkit-transition: all 0.25s;
    }

    .search input[type="text"]:visited,
    .search input[type="text"]:focus,
    .search input[type="text"]:hover,
    .search input[type="text"]:active,
    .search input[type="text"]:visited + .mat-search,
    .search input[type="text"]:focus + .mat-search,
    .search input[type="text"]:hover + .mat-search,
    .search input[type="text"]:active + .mat-search,
    .search input[type="text"]:visited::placeholder,
```

```
.search input[type="text"]:focus::placeholder,
.search input[type="text"]:hover::placeholder,
.search input[type="text"]:active::placeholder {
    color: black;
    opacity: 1;
}

.search:visited::after,
.search:focus::after,
.search:active::after,
.search:focus-within::after {
    background-color: orange;
}

.section-filter {
    margin-top: 20px;
    display: block;
    position: relative;
    font-size: 16px;
}

.section-filter-row {
    margin-top: 5px;
}

.checkbox-container {
    display: block;
    position: relative;
    cursor: pointer;
    font-family: "PT Sans", sans-serif;
    line-height: 18px;
    padding-left: 24px;
    user-select: none;
    -ms-user-select: none;
    -moz-user-select: none;
    -webkit-user-select: none;
}

.checkbox-container input {
    position: relative;
    opacity: 0;
    cursor: pointer;
}

.checkmark {
    position: absolute;
    top: 0;
```

```
            left: 0;
            height: 14px;
            width: 14px;
            background-color: white;
            border: 2px solid gray;
            transition: all 0.25s;
            -o-transition: all 0.25s;
            -moz-transition: all 0.25s;
            -webkit-transition: all 0.25s;
        }

        .checkbox-container:hover input ~ .checkmark {
            background-color: white;
            border-color: gray;
        }

        .checkbox-container input:checked ~ .checkmark {
            background-color: orange;
            border: solid 2px orange;
        }

        .checkmark:after {
            content: "";
            position: absolute;
            display: none;
        }

        .checkbox-container input:checked ~ .checkmark:after {
            display: block;
        }

        .checkbox-container .checkmark:after {
            margin-top: -0.5px;
            margin-left: 3.5px;
            width: 4.8px;
            height: 9.8px;
            border: solid white;
            border-width: 0 2px 2px 0;
            -webkit-transform: rotate(45deg);
            -ms-transform: rotate(45deg);
            transform: rotate(45deg);
        }
    }

    .list-container-filter-open {
        height: calc(100% - 360px);
    }
```

```
.list-container-filter-closed {
    height: calc(100% - 184px);
}

.list-container-filter-open,
.list-container-filter-closed {
    .list {
        position: relative;
        height: 100%;
        top: 0;
        left: 0;
        overflow-y: auto;

        &::-webkit-scrollbar {
            background-color: white;
            width: 10px;
        }

        &::-webkit-scrollbar-thumb {
            background-color: lightgray;
        }

        .category-name {
            font-size: 16px;
            font-family: "PT Sans", sans-serif;
            text-transform: uppercase;
            color: gray;
            line-height: 20px;
            padding-bottom: 15px;
            cursor: pointer;
            transition: all 0.25s;
            -o-transition: all 0.25s;
            -moz-transition: all 0.25s;
            -webkit-transition: all 0.25s;
        }

        .category-name-active,
        .category-name:hover {
            color: black;
        }

        .components {
            padding-bottom: 30px;
        }

        .component {
```

```
                font-size: 16px;
                font-family: "PT Sans", sans-serif;
                color: black;
                line-height: 25px;
                text-decoration: none;
                cursor: pointer;
                transition: all 0.25s;
                -o-transition: all 0.25s;
                -moz-transition: all 0.25s;
                -webkit-transition: all 0.25s;
            }

            .component:focus,
            .component:active,
            .component:visited {
                outline: none;
            }

            .component:hover {
                text-decoration: underline;
            }

            .active-component {
                color: orange;
            }
        }
    }
}
```

Příloha č. 4: Konstanty

documentation-items.ts

```
export interface DocItem {
    id: string;
    name: string;
    sectionId?: string;
    categoryId?: string;
}

export interface DocCategory {
    id: string;
    name: string;
    items: DocItem[];
}

export interface DOCS {
    [key: string]: DocCategory[];
}

const DIRECTIVES = 'directives';

const DOCS: DOCS = {
    [DIRECTIVES]: [
        {
            id: 'controls',
            name: 'Controls',
            items: [
                {
                    id: 'lg-button',
                    name: 'lg-button',
                },
            ],
        },
    ],
};

for ( const category of DOCS[DIRECTIVES] ) {
    for ( const item of category.items ) {
        item.sectionId = 'directives';
        item.categoryId = category.id;
    }
}

const ALL_DIRECTIVES = DOCS[DIRECTIVES].reduce(
    ( result, directives ) => result.concat( directives.items ), [] );
```

```
const ALL_ITEMS = ALL_DIRECTIVES.concat( [] );
const ALL_CATEGORIES = DOCS[DIRECTIVES].concat( [] );

@Injectable()
export class DocumentationItems {
    getDocs(): DOCS {
        return DOCS;
    }

    getAllCategories(): DocCategory[] {
        return ALL_CATEGORIES;
    }

    getCategoryById( id: string ): DocCategory {
        return ALL_CATEGORIES.find( category => category.id === id );
    }

    getAllItems(): DocItem[] {
        return ALL_ITEMS;
    }

    getItemsBySection( section: string ): DocItem[] {
        switch ( section ) {
            case DIRECTIVES:
                return ALL_DIRECTIVES;
        }
        return [];
    }

    getItemsById( id: string ): DocItem[] {
        return ALL_ITEMS.filter( item => item.id.includes( id ) );
    }

    getItemById( id: string ): DocItem {
        return ALL_ITEMS.find( doc => doc.id === id );
    }
}
```

all-components.ts

```
export const ALL_ENTRY_COMPONENTS: any[] = [
    ...directivesEntryComponents,
];
```

```
export const ALL_COMPONENTS: IEntryComponentsDefinition = {
    ...DIRECTIVES_COMPONENTS,
};
```

directives.ts

```
export const directivesEntryComponents: any[] = [
    ButtonOverviewComponent,
    ButtonAPIComponent,
]

const controlComponents: IEntryComponentsDefinition = {
    'lg-button': {
        overviewComponent: ButtonOverviewComponent,
        apiComponent: ButtonAPIComponent
    },
}

export const DIRECTIVES_COMPONENTS: IEntryComponentsDefinition = {
    ...controlComponents,
}
```

application-interfaces.ts

```
export interface IComponentDefinition {
    overviewComponent: any;
    apiComponent: any;
}

export interface IEntryComponentsDefinition {
    [key: string]: IComponentDefinition
}

export interface IAPIItem {
    name: string,
    description?: string,
}

export const API = 'api';
export const CLASS = 'class';
export const METHODS = 'methods';
export const OPTIONS = 'options';
```

Příloha č. 5: Úvodní stránka

doc-page.component.ts

```
@Component( {
    selector: 'doc-page',
    templateUrl: './doc-page.component.html'
} )
export class PageComponent { }
```

doc-page.component.html

```
<div class="header-wrapper">
    <span class="landing-page-title"> Tools </span>
    <span class="landing-page-tool-title"> Documentation </span>
</div>
<div class="landing-page">
    <h1> Welcome to the Documentation!</h1>
    <div class="text-block">
        <p> Lorem ipsum dolor sit amet consectetur adipisicing elit. </p>
        <ul>
            <li>
                <span class="keyword">Overview</span>
            </li>
            <li>
                <span class="keyword">API</span>
            </li>
        </ul>
    </div>
    <div class="text-block">
        <h2> Overview </h2>
        <p> Lorem ipsum dolor sit, amet consectetur adipisicing elit.
Magni eos doloribus necessitatibus sapiente atque
            eveniet laudantium, perspiciatis placeat nam ad corporis est
totam tempora excepturi reprehenderit odio
            praesentium, natus debitis.
        </p>
    </div>
    <div class="text-block">
        <h2> API </h2>
        <p> Lorem ipsum dolor sit amet consectetur adipisicing elit. </p>
    </div>
</div>
```

doc-page.title.ts

```
@Injectable()
export class ComponentPageTitle {
    _title = '';
    get title(): string { return this._title; }
    set title( title: string ) {
        this._title = title;
        if ( title !== '' ) {
            title = `${ title } | `;
        }
        this.bodyTitle.setTitle( `${ title }LOGEX FRAMEWORK` );
    }
    constructor( private bodyTitle: Title ) { }
}
```

Příloha č. 6: Tělo aplikace

doc-section-viewer.ts

```
@Component( {
    selector: 'doc-section-viewer',
    template: `
    <router-outlet></router-outlet>
    `
} )
export class SectionViewer { }
```

doc-category-viewer.ts

```
@Component( {
    selector: 'doc-category-viewer',
    template: `
    <router-outlet></router-outlet>
    `
} )
export class CategoryViewer { }
```

doc-component-viewer.ts

```
@Component( {
    selector: 'doc-component-viewer',
    template: `
    <router-outlet></router-outlet>
    `
} )
export class ComponentViewer {}
```

Příloha č. 7: TabViewer

doc-tab-viewer.ts

```
@Component( {
    selector: 'doc-tab-viewer',
    templateUrl: './doc-tab-viewer.html',
    styleUrls: ['./doc-tab-viewer.scss'],
} )

export class TabViewer implements OnInit, OnDestroy {
    @ViewChild( 'portal' ) portal: ElementRef;
    docItem: DocItem;
    sub: Subscription;
    currentComponent: string;
    currentCategory: string;

    private _currentTab: string;
    private _portalHost: DomPortalHost;
    private _destroyed = new Subject();

    constructor(
        private _appRef: ApplicationRef,
        private _componentFactoryResolver: ComponentFactoryResolver,
        private _injector: Injector,
        private _viewContainerRef: ViewContainerRef,
        private _docItems: DocumentationItems,
        private _title: ComponentPageTitle,
        private _route: ActivatedRoute
    ) { }

    ngOnInit() {
        combineLatest( this._route.params, this._route.parent.params,
this._route.parent.parent.params ).pipe(
            map( ( page: [Params, Params, Params] ) => ( { tab:
page[0]['tab'], id: page[1]['id'], category: page[2]['category'] } ) ),
            map( page => ( {
                tab: page.tab,
                id: page.id,
                category: page.category
            } ),
                takeUntil( this._destroyed ) )
        ).subscribe( d => this._onRouteChange( d ) );
    }

    private _onRouteChange( d: { tab: any; id: any; category: any; } ) {
        this._currentTab = d.tab;
```

```
        this.currentComponent = d.id;
        this.currentCategory = d.category;
        this.docItem = this._docItems.getItemById( d.id );
        this._title.title = `${ this.docItem.name }`;
        this._loadComponents();
    }

    private _loadComponents() {
        if ( this._portalHost && this._portalHost.hasAttached() ) {
            this._portalHost.detach();
        }

        const element = this.portal.nativeElement;
        let component: any;
        switch ( this._currentTab ) {
            case 'overview':
                component =
ALL_COMPONENTS[this.currentComponent].overviewComponent;
                break;
            case 'api':
                component =
ALL_COMPONENTS[this.currentComponent].apiComponent;
                break;
        }

        this._portalHost = new DomPortalHost( element,
this._componentFactoryResolver, this._appRef, this._injector );
        const portal = new ComponentPortal( component,
this._viewContainerRef );
        this._portalHost.attach( portal );
    }

    ngOnDestroy(): void {
        this._portalHost.detach();
        this._destroyed.next();
    }
}
```

doc-tab-viewer.html

```
<div class="header-wrapper">
    <span class="category-title">
        {{currentCategory}}
    </span>
    <span class="component-title">
```

```
        {{currentComponent}}
    </span> </div>
<div class="tab-wrapper">
    <ul>
        <li routerLink="../overview" routerLinkActive="active-route">
            <span>
                <b>
                    Overview
                </b>
            </span> </li>
        <li routerLink="../api" routerLinkActive="active-route">
            <span>
                <b>
                    API
                </b>
            </span> </li>
    </ul>
</div>
<div #portal> </div>
```

doc-tab-viewer.scss

```
.tab-wrapper {
    margin-top: 30px;
    margin-bottom: 70px;

    ul {
        list-style-type: none;

        li {
            display: inline-block;
            margin-left: 60px;
            cursor: pointer;

            span {
                font-family: "PT Sans", sans-serif;
                font-size: 16px;
                line-height: 20px;
                color: orange;
                text-transform: uppercase;
                transition: all 0.25s;
                -o-transition: all 0.25s;
                -moz-transition: all 0.25s;
                -webkit-transition: all 0.25s;
            }
```

```
            &:visited,
            &:focus,
            &:active,
            .tab-wrapper ul li {
                outline: none;
            }

            &:first-child {
                margin-left: 0px;
            }

            &:after {
                display: block;
                left: 0;
                margin-top: 4px;
                content: "";
                height: 3px;
                width: 0px;
                background-color: blue;
                transition: all 0.25s;
                -o-transition: all 0.25s;
                -moz-transition: all 0.25s;
                -webkit-transition: all 0.25s;
            }

            &:hover::after {
                width: 100% !important;
            }
        }
        .active-route {
            span {
                color: orange !important;
            }
            &::after {
                width: 100% !important;
            }
        }
    }
}
```

Příloha č. 8: Automatizace API záložky

api-viewer-base.ts

```
export type ApiDefinition = {
    [key: string]: IAPIItem[]
};

@Injectable()
export class ApiViewerBase implements AfterViewInit {
    protected _definition: ApiDefinition;
    protected _selector: string;

    protected _sections: string[] | ( () => string[] );

    constructor(
        private _changeDetectorRef: ChangeDetectorRef
    ) { }

    ngAfterViewInit(): void {
        this._sections = this._getAllSections();
        if ( this._selector === undefined || this._selector.length < 1 )
{
            this._selector = 'There is no selector for this component';
        }
        this._changeDetectorRef.detectChanges();
    }

    private _getAllSections(): string[] {
        return Object.keys( this._definition );
    }
}
```

api-viewer-base.component.html

```
<div class="text-block">
    <h2>Directive</h2>
    <p> Selector: {{_selector}} </p>
</div>
<div *ngFor="let section of _sections">
    <api-table-viewer [definition]="_definition[section]"
[current]="section"></api-table-viewer>
</div>
```

api-table-viewer.component.ts

```
export type ApiDefinition = { [key: string]: IAPIItem[] };

@Component( {
    selector: 'api-table-viewer',
    templateUrl: './api-table-viewer.component.html'
} )
export class ApiTableViewer implements AfterViewInit {
    @Input( 'definition' ) _apiItems: IAPIItem[];
    @Input( 'current' ) _apiItemName: string;

    _hasDescription: boolean;

    constructor(
        private _changeDetectorRef: ChangeDetectorRef,
    ) {
        this._hasDescription = true;
    }

    getName( id: number ): string {
        return this._apiItems[id].name;
    }

    getDesc( id: number ): string {
        return this._apiItems[id].description;
    }

    ngAfterViewInit() {
        this._apiItemName = this._apiItemName.toUpperCase();
        this._apiItems.map( item => {
            if ( !item.description ) {
                this._hasDescription = false;
            }
        } );
        this._changeDetectorRef.detectChanges();
    }
}
```

api-table-viewer.component.html

```
<div *ngIf="_apiItems.length != 0">
    <h3>{{_apiItemName}}</h3>
    <table class="table table-striped">
        <thead class="thead-light">
            <tr>
```

```
                <th>Attribute name</th>
                <th *ngIf="_hasDescription">Description</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let item of _apiItems, index as i">
                <td>{{getName(i)}}</td>
                <td *ngIf="_hasDescription">{{getDesc(i)}}</td>
            </tr>
        </tbody>
    </table>
</div>
```

Příloha č. 9: ExampleViewer

example-viewer.ts

```
@Directive({
    selector: '[fileName]'
})
export class FileName {
    @Input('fileName') fileName: string;
}

@Component({
    selector: 'example-viewer',
    templateUrl: './example-viewer.html',
    styleUrls: ['./example-viewer.scss']

})
export class ExampleViewer implements AfterContentInit, AfterViewInit {
    @ContentChild('comment') commentElement: ElementRef;
    @ContentChild(FileName) fileNameDirective: FileName;
    @ContentChild('template') exampleTemplate: TemplateRef<any>;
    @ViewChild('exampleContainer', { read: ViewContainerRef })
exampleContainer: ViewContainerRef;
    _isCodeVisible: boolean;
    _fileName: string;
    _comment: string;

    constructor(
        private _changeDetectorRef: ChangeDetectorRef,
    ) { }

    ngAfterContentInit(): void {
        this._comment = this.commentElement.nativeElement.textContent;
        this._fileName = this.fileNameDirective.fileName;
    }

    ngAfterViewInit(): void {
        this._fileName = this.fileNameDirective.fileName;
        this.exampleContainer.createEmbeddedView(this.exampleTemplate);
        this._changeDetectorRef.detectChanges();
    }

    _getCodeVisibility(visible: boolean): void {
        this._isCodeVisible = visible;
    }
}
```

example-viewer.html

```
<div class="example-wrapper">
    <div class="example-block">
        <example-code-viewer
            (clicked)="_getCodeVisibility($event)"
            [fileName]="_fileName"
        ></example-code-viewer>
        <div
            class="example-component"
            [class.example-component-pushed]="_isCodeVisible"
        >
            <div #exampleContainer> </div>
        </div>
        <div class="row"></div>
    </div>
    <div class="example-comment"> {{_comment}} </div>
</div>
```

example-viewer.scss

```
.example-wrapper {
    margin-bottom: 90px;
    margin-top: 30px;
    .example-block {
        position: relative;
        width: 100%;
        background-color: lightgray;
        padding: 20px;
        margin-left: -20px;

        .example-component {
            position: relative;
            width: 90%;
            float: left;

            &.example-component-pushed {
                padding-top: 20px;
            }
        }
    }

    .example-comment {
        margin-top: 10px;
        color: lightgray;
        font-family: "PT Sans", sans-serif;
        line-height: 25px;
    }
}
```

Příloha č. 10: ExampleCodeViewer

example-code-viewer.ts

```
@Component({
    selector: 'example-code-viewer',
    templateUrl: './example-code-viewer.html',
    styleUrls: ['./example-code-viewer.scss'],
    encapsulation: ViewEncapsulation.None,
})

export class ExampleCodeViewer implements OnDestroy {
    @ViewChild('htmlBtn') htmlBtn: ElementRef;
    @ViewChild('tsBtn') tsBtn: ElementRef;
    @Input('fileName') fileName: string;
    @Output('clicked') isCodeButtonClicked: EventEmitter<boolean> = new
EventEmitter();
    _isCodeVisible: boolean;
    _isHtmlVisible: boolean;
    _isTsVisible: boolean;
    _code: string;
    private readonly _destroyed$ = new Subject<void>();

    constructor(
        private _router: Router,
        private _httpClient: HttpClient
    ) {
        this._isCodeVisible = false;
        this._isHtmlVisible = false;
        this._isTsVisible = false;
    }

    toggleHeader() {
        this._isCodeVisible = !this._isCodeVisible;
        if (this._isCodeVisible) {
            this.toggleHtml();
        }
        this.isCodeButtonClicked.emit(this._isCodeVisible);
    }

    toggleHtml() {
        this._isHtmlVisible = true;
        this._isTsVisible = false;
        this._getUrl('html');
    }

    toggleTs() {
```

```
        this._isTsVisible = true;
        this._isHtmlVisible = false;
        this._getUrl('ts');
    }

    private _getUrl(fileExtension: string): void {
        const urlParts = this._router.url.split('/');
        urlParts.pop();
        const folderPath = urlParts.join('/');
        const filePath = this._getFileUrl(folderPath, this.fileName,
fileExtension);
        this._httpClient
            .get(filePath, { responseType: 'text' })
            .pipe(takeUntil(this._destroyed$))
            .subscribe(document => {
                this._code = this._getCode(document);
            });
    }

    private _getFileUrl(path: string, fileName: string, fileExtension:
string): string {
        return `assets/examples/${path}/${fileName}-
${fileExtension}.html`.toLowerCase();
    }

    private _getCode(doc: string): string {
        if (!doc) return 'The example has no code available.';
        return doc;
    }

    ngOnDestroy(): void {
        this._destroyed$.next();
        this._destroyed$.complete();
    }
}
```

example-code-viewer.html

```
<div class="row">
    <button
        type="button"
        #showBtn
        class="show-code-btn"
        (click)="toggleHeader()"
        [class.show-code-btn-active]="_isCodeVisible"
```

```
    >
        <i class="material-icons"> code </i>
    </button>
    <div *ngIf="_isCodeVisible">
        <button
            type="button"
            #tsBtn
            class="show-language-btn"
            (click)="toggleTs()"
            [class.show-language-btn-active]="_isTsVisible"
        >
            <b> TS </b>
        </button>
        <button
            type="button"
            #htmlBtn
            class="show-language-btn"
            (click)="toggleHtml()"
            [class.show-language-btn-active]="_isHtmlVisible"
        >
            <b> HTML </b>
        </button>
    </div>
</div>
<div
    *ngIf="_isCodeVisible"
    class="row"
>
    <div>
        <pre>
<code highlight
[textContent]="_code"
>
</code>
    </pre> </div>
</div>
```

example-code-viewer.scss

```
.show-code-btn {
    display: block;
    height: 30px;
    font-size: 16px;
    line-height: 15px;
    margin-left: 30px;
    float: right;
    background-color: transparent;
    border: none;
```

```
    text-align: center;
    cursor: pointer;
    position: relative;
    width: 30px;
    color: gray;
    outline: none;

    i {
        font-size: 20px;
    }

    &:visited,
    &:focus,
    &:active {
        position: relative;
        width: 30px;
        color: gray;
        outline: none;
    }

    &:hover,
    &.show-code-btn-active,
    &.show-code-btn-active:visited,
    &.show-code-btn-active:focus,
    &.show-code-btn-active:active {
        color: white !important;
        background-color: #333333;
        outline: none;
    }
}

.show-language-btn {
    display: block;
    height: 30px;
    font-size: 16px;
    line-height: 15px;
    margin-left: 30px;
    float: right;
    background-color: transparent;
    border: none;
    text-align: center;
    cursor: pointer;
    font-family: "PT Sans", sans-serif;
    color: gray;
    transition: all 0.25s;
    -o-transition: all 0.25s;
    -moz-transition: all 0.25s;
```

```
    -webkit-transition: all 0.25s;
    outline: none;

    &:visited,
    &:focus,
    &:active {
        font-family: "PT Sans", sans-serif;
        color: gray;
        transition: all 0.25s;
        -o-transition: all 0.25s;
        -moz-transition: all 0.25s;
        -webkit-transition: all 0.25s;
        outline: none;
    }

    &.show-language-btn-active {
        color: black !important;
        outline: none;
        &::after {
            display: block;
            left: 0;
            margin-top: 1px;
            content: "";
            height: 3px;
            width: 100%;
            background-color: orange;
        }
    }

    &::after {
        display: block;
        left: 0;
        margin-top: 1px;
        content: "";
        height: 3px;
        width: 0px;
        background-color: orange;
        transition: all 0.25s;
        -o-transition: all 0.25s;
        -moz-transition: all 0.25s;
        -webkit-transition: all 0.25s;
    }

    &:hover::after {
        width: 100%;
    }
}
```

Příloha č. 11: Přidání komponenty

example-overview.component.html

```
<div class="tab-content">
    <div class="text-block">
        Lorem ipsum dolor sit amet consectetur adipisicing elit.
    </div>
    <example-viewer>
        <ng-template #template>
            <div>
                First example
            </div>
        </ng-template>
        <div #comment> Example 1 </div>
        <div fileName="example-1"></div>
    </example-viewer>
    Lorem ipsum dolor sit amet, consectetur adipisicing elit.
</div>
```

example-overview.component.ts

```
@Component( {
    selector: 'example-overview',
    templateUrl: './example-overview.component.html',
} )
export class ExampleComponent {
}
```

example-api.component.ts

```
@Component( {
    selector: 'example-api',
    templateUrl: '../../../../../api-viewer-base/api-viewer-
base.component.html'
} )
export class ExampleAPIComponent extends ApiViewerBase implements OnInit
{
    ngOnInit(): void {
        this._definition = ExampleAPIContents;
        this._selector = '<example>';
    }
}
```

example-api.ts

```
export const ExampleAPIContents: { [key: string]: IAPIItem[] } = {
    [API]: [
        {
            name: 'Parameter 1',
            description: 'Description 1.'
        },
```

```
        {
            name: 'Parameter 2',
            description: 'Description 2.'
        }
    ]
}
```

example-1-html.html

```
<div>
    First example
</div>
```

example-1-ts.html

```
```